Metropolia Ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Kimmo Niittynen

Videon tuottamiseen ja lähetykseen käytettävän älypuhelinsovelluksen suunnittelu ja toteutus

Insinöörityö 10.5.2009

Ohjaaja: kehitysjohtaja Kari Melkko
Ohjaava opettaja: yliopettaja Hannu Laine

Metropolia Ammattikorkeakoulu **Insinöörityön tiivistelmä**

Tekijä Otsikko Sivumäärä Aika	Kimmo Niittynen Videon tuottamiseen ja lähetykseen käytettävän älypuhelinsovelluksen suunnittelu ja toteutus 80 sivua 10.5.2009
Koulutusohjelma	tietotekniikka
Tutkinto	insinööri (AMK)
Ohjaaja Ohjaava opettaja	kehitysjohtaja Kari Melkko yliopettaja Hannu Laine

Insinöörityössä tehtävänä oli suunnitella ja toteuttaa matkapuhelimessa toimiva sovellusohjelma, jonka avulla voi kuvata videota. Kuvattu video tuli voida lähettää palvelimelle, joka tallentaa videon opetusvideopankkiin katseltavaksi. Helppokäyttöisyys oli sovelluksen tärkein vaatimus.

Työ tehtiin Mamentor Oy:lle laajennuksena olemassa olevan MentorAid-tuotteen ominaisuuksiin. MentorAid on selainkäyttöinen osaamisen ja osaamispääoman hallintajärjestelmä, jonka yhtenä käyttötarkoituksena on tarjota työkaluja erilaisten kurssien läpivientiin. Etäkurssit koostuvat pääosin videomateriaalista ja mahdollisista osaamiskartoituksista. Asiakasyritykset tuottavat opetusvideoita omissa tiloissaan tai Mamentor Oy:n videostudiossa. Videoita katselevat etäkursseilla opiskelevat asiakasyritysten työntekijät tai yhteistyökumppanit. Nyt suunniteltavan sovelluksen tarkoitus on madaltaa opetusvideoiden tuottamisen kynnystä.

Sovellus toteutettiin Symbian C++ -ohjelmointikielellä S60 3rd Edition -alustalle. Käyttöliittymä ohjelmoitiin käyttäen S60-näkymänvaihtoarkkitehtuuria (view switching). Multimediakehysrakenne (MMF) ja HTTP-kehysrakenne olivat käytössä videon kuvaamisessa ja siirrossa.

Lopullinen tuote jäi keskeneräiseksi, koska projekti keskeytettiin taloudellisen tilanteen pakottamana. Tuotteeseen toteutettiin käyttöliittymä kokonaisuudessaan noudattaen korkeaa käytettävyysvaatimusta. Tiedonsiirtoyhteys toteutettiin valmiiksi ja testattiin. Videokuvan tallentaminen on kirjoitettu, mutta ei läpikotaisin testattu. Testausta vaadittaisiin myös koko sovellukselle, mikäli se aiottaisiin ottaa tuotantoon. Kuvatun videon katselu suunniteltiin myös, mutta sen toteutus jäi odottamaan projektin mahdollista jatkumista.

Hakusanat	Symbian, mobiili, sovellus, S60, video, HTTP

Helsinki Metropolia University of Applied Sciences **Abstract**

Author Title Number of Pages Date	Kimmo Niittynen Smartphone application for recording and uploading videos: design and implementation 80 pages 10 May 2009
Degree Programme	Information Technology
Degree	Bachelor of Engineering
Instructor Supervisor	Kari Melkko, Chief Development Officer Hannu Laine, Principal Lecturer

The purpose of this thesis was to design and implement a video recording application for mobile phones. The recorded video was also to be submitted to a server to be stored in a tutorial video database. Usability was a key requirement for the application.

The work was done for Mamentor Oy as an extension feature for an existing software product, MentorAid. MentorAid is a skill and knowledge management solution used via a web browser. One of the main purposes of the software is to provide tools for arranging online courses. The online courses mainly consist of a variety of video materials and questionnaires. Mamentor Oy hosts a video studio that clients can use to record their educational material. The educational videos are viewed by course attendees, who usually are employees or affiliates of the client in question. The smart phone software should make creating videos more attractive by reducing the complexity of the recording process.

The application was implemented with Symbian C++ on S60 3rd Edition platform. The user interface was programmed using S60 view switching architecture. Multimedia framework and HTTP-framework were used to record and send the video files. Persistence was implemented to save the application state when exiting and to restore it on application launch.

The project was put on hold due to financial concerns in the final phases of the development. This led to problems with completing the prototype. The user interface was implemented to meet the high usability criteria. Data transfer functionalities were fully programmed. Video recording is partially unfinished, as the code has been written but proper testing has not been performed yet. If the project was to continue, systematic testing would have to be carried out on the whole application. Viewing of recorded videos was designed, but the implementation was left for the possible continuation of the project.

Keywords	Symbian, mobile, application, S60, video, HTTP

Sisällys

Lyhenne- ja käsiteluettelo

API	Application Programming Interface; sovellusohjelmointiin tarkoitettu ohjelmointirajapinta
Asynkroninen operaatio	Operaatio, jonka valmistumista ei jäädä odottamaan. Operaation valmistumisesta annetaan tieto sen tarkkailijalle.
CVS	Concurrent Versions System; ohjelmistonkehityksessä käytettävä versionhallintajärjestelmä
Erottelukyky	Resoluutio; kuvan muodostavien kuvapisteiden määrä (pinta-alayksikköä kohti)
Flash-video	Tiedostoformaatti, jota käytetään videon jakamiseen Internetissä käyttäen Adobe Flash Player –sovellusta
GPS	Global Positioning System; yhdysvaltalaisessa omistuksessa oleva maailmanlaajuinen satelliittipaikannusjärjestelmä [1.]
HTML	Hypertext Markup Language; hypertekstin kuvauskieli; Käytetään esimerkiksi WWW-sivujen määrittelyyn
HTTP	Hypertext Transfer Protocol; hypertekstin siirtoprotokolla; HTTP-protokollaa käytetään yleisesti tiedonsiirtoon Internetissä.
HTTP-istunto	Tilattoman HTTP-protokollan yhteyden ylläpidon väline, joka toteutetaan usein tallentamalla asiakasohjelmaan evästeitä.
HTTPS	Hypertext Transfer Protocol Secure; salattu versio HTTP-protokollasta
H.264	Videon pakkaus- ja koodausstandardi, osa MPEG-4-perhettä [2, s. 1; 3, s. 1.]
Java ME	Java Micro Edition; Java-ohjelmointikielen ominaisuuksiltaan rajoitettuihin ympäristöihin tarkoitettu versio
JSP	Java Server Pages; Java-ohjelmointikielen dynaamiseen Web-sivujen luomiseen kehitetty tekniikka [4.]
Kyvykkyys	Capability; Symbian-käyttöjärjestelmän käyttämä käyttöoikeuksien perusyksikkö
MPEG-4	Perhe avoimia, kansainvälisiä standardeja, joka tarjoaa työkaluja multimedian jakamiseen [3, s. 1].

Kehysrakenne	Kirjasto tietyn aihealueen ohjelmointia helpottavia valmiita ohjelmakomponentteja
PyS60	Python for S60; S60-alustalle tuotu versio Python-ohjelmointikielestä [5.]
RFID	Radio Frequency Identification; yleisnimitys radiotaajuuksilla toimiville etätunnistustekniikoille [6.]
SAX	Simple API for XML; yksinkertainen, sarjamuotoinen tapa lukea XML-tiedostoja
SDK	Software Development Kit; valikoima ohjelmistonkehitystyökaluja tietyllä kielellä tapahtuvaan kehitykseen
Symbian C++	Symbian OS-käyttöjärjestelmän ohjelmointikieli
Symbian OS	Pienitehoisille laitteille suunniteltu käyttöjärjestelmä. Käytössä laajalti matkapuhelimissa [7.]
S60	Nokian kehittämä, Symbian OS-käyttöjärjestelmän päälle rakennettu sovellusalusta matkapuhelimille
TCP	Transmission Control Protocol; kuljetuskerroksen protokolla, jolla luodaan yhteyksiä tietokoneiden välille [8.]
UML	Unified Modeling Language; ohjelmistosuunnittelussa usein käytetty standardoitu, graafinen mallinnuskieli
URL	Uniform Resource Locator; tapa osoittaa resurssin paikka. Käytetään esimerkiksi Internet-resurssien osoittamiseen
Välitön etsintila	Direct viewfinder mode; kameran etsimen tila, jossa kuva muodostetaan näytölle suoraan muistista, eikä väliaikaisesta bittikarttakuvasta [9, s. 44-45.]
WLAN	Wireless Local Area Network; langaton lähiverkko
XML	Extensible Markup Language; yksinkertainen ja joustava tekstipohjainen tapa kuvata tietoa, jota käytetään esimerkiksi tiedon välitykseen Internetissä. [10.]

1 Johdanto

Mamentor Oy on perustettu vuonna 2006, ja aluksi sen liiketoiminta koostui tietoliikennealan koulutuskokonaisuuksien järjestämisestä ja konsultoinnista. Yritys on kasvanut nopeasti, ja vuonna 2008 sen palveluksessa toimi 13 työntekijää. Oman MentorAid-tuotteen suunnittelu ja kehittäminen aloitettiin, kun havaittiin markkinoilla vallitseva pula kunnollisista selainkäyttöisistä oppimis- ja osaamisenhallintaratkaisuista. Nykyään MentorAidin myynti on merkittävä osa yrityksen liiketoimintaa.

MentorAid-osaamisenhallintajärjestelmä on suunnattu avuksi erilaisten yritysten osaamisenhallinnan haasteisiin. Järjestelmää myydään räätälöitynä palvelupakettina, johon kuuluu asennus- ja tukitoimintojen lisäksi asiakkaan tilauksen mukaisia ohjelmistokomponentteja. Näitä ohjelmistokomponentteja ovat osaamisen kehitys-, kartoitus- ja hallintatoiminnot sekä virtuaalinen tietoverkkolaboratorio. Tuotetta käytetään asiakasyrityksissä usein sisäiseen koulutukseen. Osaamisen kehityksen välineenä toimivat usein kurssit, jotka koostuvat pääosin videomateriaaleista. Opetusvideomateriaalin kuvaamiseen on asiakkaan halutessaan mahdollista käyttää Mamentor Oy:n tiloissa sijaitsevaa, tarkoitukseen suunniteltua kuvausstudiota.

Tässä insinöörityössä esitelty tuote on tehty Mamentor Oy:lle laajennuksena olemassa olevaan MentorAid-osaamisenhallintajärjestelmään. Mahdollinen tarve järjestelmän laajenemiselle älypuhelimiin ilmeni erään asiakkaan pyynnöstä. Tähän asti MentorAid-järjestelmässä julkaistavia opetusvideoita on kuvattu lähinnä asiakasyritysten omissa tiloissa tai Mamentor Oy:n videostudiossa. Tämä menetelmä toimii suhteellisen hyvin tietoteknisten toimintojen suorittamista kuvattaessa. Asiakkaalla oli kuitenkin tarve saada kuvattua käytännön tilanteita työolosuhteissa ilman mittavaa kuvauskalustoa tai studiota. Kuvattavat kohteet voivat myös olla luonteeltaan sellaisia ettei niitä voi, tai ole järkevää siirtää. Opetusvideon kuvaamisen tulisi olla helppoa ja suoraviivaista, ja tarvittavien laitteiden tulisi kulkea aina mukana. Asiakkaan vaatimuksia tarkennettiin hieman tuotanto- ja suunnittelutiimin voimin, ja päädyttiin seuraavaan yleisluontoiseen vaatimusten kuvaukseen. Tarvittaisiin älypuhelinsovellus, jolla on mahdollista kuvata videokuvaa ja lähettää se opastevideoksi MentorAid-järjestelmään. Videoon tulisi voida myös liittää nimi, videon kuvaus, kuvauspaikka sekä joitakin osaamisen hallintaan liit-

tyviä lisämääreitä. Lopullisen tuotteen pitäisi ehdottomasti olla erittäin helppokäyttöinen, sillä sitä pitäisi voida käyttää myös hyvin vähäisillä tietoteknisillä taidoilla. Tällainen sovellus tuntui sopivan yrityksen tulevaisuuden suunnitelmiin. Pian päätettiinkin, että on paikallaan tehdä aiheesta käytännön tutkimusta suunnittelemalla matkapuhelinsovellus ja tekemällä siitä prototyyppi esiteltäväksi asiakkaille sekä yrityksen sisäiseen arviointiin päätöksenteon tueksi.

Työssä kuvaillaan sovelluskehitysprosessin vaiheet ideasta suunnitteluun ja toteutukseen, tehdyt valinnat ja ratkaisut eri työvaiheissa sekä työn aikana suunnitelmaan tehdyt muutokset. Lisäksi arvioidaan vielä jatkokehityksen hyötyjä ja ongelmia sekä sovelluksen jatkokehityksen kannattavuutta. Tekstissä keskitytään päätelaitteeseen asennettavan ohjelman suunnitteluun ja toteutukseen, mutta esitellään myös lyhyesti, mitä muutoksia olemassa olevaan selainkäyttöiseen järjestelmään joudutaan tekemään.

2 Huomioita älypuhelinsovelluksista

2.1 Alustan rajoitukset

Älypuhelimet lähestyvät teknisesti monessa suhteessa työasemakoneita. Joillakin osa-alueilla niissä on kuitenkin rajoituksia, joista ei pääse eroon. Älypuhelimet toimivat akkukäyttöisesti, mikä johtaa tiukkoihin virransäästövaatimuksiin. Tämä tarkoittaa sitä, että esimerkiksi taustavalo ja kamera on sammutettava viiveellä aina kun puhelinta ei käytetä. Ohjelmistokehittäjälle koituu muitakin vaatimuksia virransäästöstä. Algoritmit on aina optimoitava eikä ylimääräisten operaatioiden suoda kuluttavan virtaa tai suorittimen rajallista kapasiteettia. Tavanomaisia matkapuhelinsovelluksia ei pienempi suoritinteho juurikaan rajoita, mutta se tulee silti ottaa huomioon suunnittelussa. Muistia on selvästi vähemmän kuin työasemissa, ja sen käyttö on harkittava tarkkaan. On noudatettava Tommi Mikkosen esittelemiä ohjenuoria, kuten "Varaa myöhään, vapauta aikaisin" ja "Varaa kaikki muisti sovelluksen käynnistyessä", jotta muisti riittäisi kaikille toivotuille operaatioille [11, s. 32–33]. Muistin koko rajoittaa myös tiedon käsittelyä. Esimerkiksi XML-tiedostojen (extensible markup language) käsittely tehdään älypuhelinlaitteissa lähes kokonaan SAX:n (simple API for XML) mukaan, lukien vain eteenpäin

tallentamatta dokumenttirakennetta väliaikaismuistiin. Tämä johtuu siitä, ettei suuri XML-dokumentti mahtuisi edes laitteen koko käyttömuistiin. Myös monet ohjelmointikielten käytännöt on suunniteltu säästämään resursseja. Esimerkiksi muistia pyritään säästämään Symbian OS -käyttöjärjestelmän oman ohjelmointikielen merkkijonojen käsittelyssä. Potentiaalisesti ongelmallinen merkkijonojen käsittely hoidetaan Symbian C++:ssa "deskriptoreilla", vaikka niitä jotkut sovelluskehittäjät kummastelevatkin. Deskriptorien tausta-ajatuksena on yhdenmukaistaa muistin käyttöä.

Näytön koko on myös selkeä rajoitus. Matkapuhelimen näytölle ei voi mahtua yhtä paljon asioita kuin työasemakoneen suurelle näytölle. Hiiren kaltaista syötelaitetta ei useimmissa matkapuhelimissa ole, joten navigointi näytön kontrolleissa on suunniteltava sujumaan näppäinpainalluksilla.

2.2 Uudenlaisia mahdollisuuksia

Rajoitusten lisäksi mobiililaitteisiin liittyy kiinteästi myös niiden olemuksen tuomia mahdollisuuksia. Mobiililaitteet on luotu kulkemaan lähes aina mukana. Tätä vahvaa läsnäoloa tulisi käyttää luomalla täysin uudenlaisia matkapuhelinsovelluksia. Miksei voisi olla sovellusta kaverinpaikannukseen tai paikannetun hätäilmoituksen tekemiseen parilla napinpainalluksella? Edellisessä esiin tuli myös eräs vapauttava syötetapa, satelliittipaikannus GPS:n (global positioning system) avulla. Paikannus tuo matkapuhelimeen mahdollisuuden näyttää tietoja paikallisesti. Yksinkertaisena esimerkkinä toiminee vaikka paikallistettu säätiedote. Melko uutena ominaisuutena puhelimissa on kiihtyvyysanturien käyttö. Antureita voisi käyttää vaikkapa eleiden käyttöön komentoina. Käytössä on jo esimerkiksi musiikin toistossa eteen- ja taaksepäin kelaus kallistamalla laitetta oikealle tai vasemmalle. Mielenkiintoisia ovat myös liikkeen tunnistus kameran avulla sekä RFID-tunnisteiden lukeminen matkapuhelimilla. Puhelimen muuntaminen valvontakamerakännykäksi tai käyntikorttien vaihto koskettamalla voisivat olla näiden tekniikoiden sovellusalueita.

2.3 Sovelluskehitys älypuhelimelle

Käyttöjärjestelmät on mobiililaitteisiin suunniteltu nämä rajoitukset ja mahdollisuudet huomioonottaen. Kehitettäessä sovelluksia älypuhelinalustalle ei jouduta muuttamaan kehitysprosessia lainkaan, kunhan nuo pienet erot huomioidaan tarvittaessa. Yksi huomioon otettavista seikoista on tasaisen hyvä käytettävyys. Hyvä käytettävyys on erittäin tärkeää matkapuhelinsovelluksissa, mutta samalla vaikea saada toteutumaan [11, s. 62]. Näytön koko on myös huomionarvoinen ongelma. Kaikki haluttavat kontrollit eivät välttämättä mahdu näytölle, ja onkin karsittava näytettävän tiedon määrää. Käyttöliittymän suunnitteleminen helppokäyttöiseksi mutta informatiiviseksi onkin hyvin haastavaa.

3 Vaatimuksia tuotteelle

3.1 Toiminnallisuus

Matkapuhelinsovelluksen toiminta tulee suunnitella siten, että käyttäjän on helppo suorittaa tarvittavat toiminnot oikeassa järjestyksessä. Sovelluksen käynnistyttyä tulee videon kuvaamiseen siirtymisen olla selvästi näkyvillä. Kuvausnäkymässä on näkymästä poistumisen lisäksi toimintona vain kuvaamisen aloittaminen. Kuvauksen käynnistämisen tulee olla helposti valittavissa yhdellä napinpainalluksella. Aloitus voidaan tehdä myös valikosta käsin. Videokuvauksen pysäyttäminen ja kuvauksen jatkaminen voidaan suorittaa samasta painikkeesta painamalla. Jokin muu painike lopettaa tallennuksen. Kuvattaessa videotiedosto tallentuu valinnaiseen tallennuspaikkaan puhelimen pysyväismuistissa. Kuvattuun videoon liitetään myös tieto kuvauspaikasta GPS-paikannuksen avulla. Jos GPS-laitetta ei ole saatavilla, kysytään kuvauspaikkatieto käyttäjältä. GPS-yksiköltä saadut koordinaatit liitetään videon tietoihin, josta ne ovat palvelimella tulkittavissa ennalta määrätyksi paikaksi. GPS-koordinaattien tai nimetyn paikan avulla voi palvelimella etsiä tiettyyn paikkaan liittyviä opetusvideoita.

Kuvaamisen jälkeen nähdään lista kuvatuista videoista. Listassa näkyville videoille voi valitsemalla määritellä nimen, lyhyen kuvauksen, sekä joitakin tunnistetietoja. Videon

tietojen syöttämisen jälkeen voi videon halutessaan esikatsella. Esikatselu tapahtuu kuvausnäkymää vastaavassa näkymässä. Videon toiston voi aloittaa valintapainikkeella tai valikkokomennolla. Videon katselun lopetettuaan voi käyttäjä poistua näkymästä.

Valintalistalta voi myös lähettää valitsemansa videon palvelimelle tallennettavaksi. Lähetys tapahtuu siirtämällä valintalistan kohdistin halutun videon kohdalle ja valitsemalla valikosta "Lähetä". Ennen varsinaisen videotiedoston lähetystä on tietoa lähettävä taho tunnistettava käyttäjätunnuksen ja salasanan avulla. Edellä mainitut tiedot kysytään käyttäjältä ja annetaan palvelimen tarkastettavaksi. Jos tiedot varmistuvat oikeiksi, videon lähetys voi alkaa. Lähettämiseen käytetään HTTP-protokollaa, eikä se saa häiritä puhelimen muuta toimintaa. Lähetyksen ollessa käynnissä on sen selkeästi näyttävä käyttäjälle. Jotta käyttäjä tietää meneillään olevasta operaatiosta, on sovelluksen jäätävä odotustilaan tiedonsiirron päättymiseen saakka. Käyttäjä voi keskeyttää tiedonsiirron milloin tahansa. Tällöin jo siirretty tieto poistetaan palvelimelta. Koska videotiedostot ovat suuria, on lähetyksen tapahduttava osissa. Palvelin kokoaa siirretyt palat uudelleen videotiedostoksi. Siirrettävän videomateriaalin vastaanottoa varten on palvelin saatava ottamaan lähetettävä video vastaan. Onnistuneesti vastaanotettu video tallennetaan tietokantaan ja sille annetaan lähetyksen mukana kulkeneet lisätiedot.

Nyt video on löydettävissä opetusvideokirjastosta annetuilla tunnistetiedoilla. MentorAid-järjestelmään kirjautunut käyttäjä voi siis katsella palvelimelle lähetetyn videon etsittyään sen vaikkapa paikkatiedon avulla. Videon toiston on onnistuttava upotettuna muuhun sisältöön ja videontoisto-ohjelma ei saa asettaa sivuston käytölle uusia vaatimuksia. Videontoistoon käytettävän apuohjelman tukemat videomuodot voivat kuitenkin asettaa lisävaatimuksia videota tallentavalle matkapuhelinsovellukselle.

Puhelimeen asennetun sovelluksen asetuksia on myös voitava muokata. Muokattaviin asetuksiin kuuluvat esimerkiksi vastaanottavan MentorAid-palvelimen osoite, mahdollinen välityspalvelin sekä videon laatuun liittyviä määreitä. Nämä asetukset sekä kaikki videoiden tiedot on tallennettava sovelluksen sulkemisen yhteydessä ja ladattava uudelleen sen käynnistyessä. Tallennus- ja lataustoimet on tehtävä käyttäjälle näkymättömästi.

Näkymäkohtaisten toimintojen lisäksi on toteutettava sovelluksenlaajuisesti valikkoon "Tietoja sovelluksesta" -kohta, jonka valitsemalla saa näytölle sovellusohjelman nimen, versionumeron sekä toimittajan. Lisäksi valikosta on näkymästä riippumatta voitava sulkea sovellus. Päänäkymästä sovelluksen voi sulkea myös pikanäppäimellä, joka muissa näkymissä aiheuttaa paluun päänäkymään. Kaikkien aikaa vievien prosessien aikana näytetään puhelimen ruudulla odotusviesti, jossa näkyy suuntaa antava jäljellä oleva suoritusaika ja tieto siitä, mitä tehdään. Virhetilanteista ja onnistuneista operaatioista on myös ilmoitettava käyttäjälle selvästi erottuvalla viestillä.

3.2 Käytettävyys

Ohjelmiston käytön on oltava helppoa ja loogista. Tommi Mikkosen [11, s. 64] huomio ”Käyttäjälle on kerrottava aina mitä laite tekee, varsinkin silloin kun ei näytä tapahtuvan mitään”, otetaan suunnittelussa tarkasti huomioon. Jos käyttäjä joutuu liikaa miettimään, mitä pitikään tehdä, tai selaamaan erilaisia valikoita, ei sovelluksen käyttö ole miellyttävää. Käytön kömpelyys johtaa siihen, ettei sovellusta käytetä. Huomioitava on myös, että testatut, hyväksytetyt ja allekirjoitetut matkapuhelinsovellukset ovat allekirjoittamattomia helpompia käyttää luotettavuuteen perustuvien suurempien toimintavaltuuksiensa myötä. Esimerkiksi tietoa siirtävät, allekirjoitetut ohjelmat kysyvät vain kerran asennuksen aikana, saavatko jatkossa käyttää tiedonsiirtopalveluita. Vastaavat allekirjoittamattomat sovellukset joutuvat kysymään käyttäjän lupaa aina tietoa siirtäessään. Käytettävyydelle asetetaan huomattavasti korkeampia vaatimuksia kuin yleensä, koska tuotettavan sovelluksen pitäisi olla käytettävissä hyvinkin vaatimattomilla tietoteknisillä taidoilla.

3.3 Tuottavuus

Lopputuotteen on tarkoitus vapauttaa opetusvideoiden kuvaaminen tapahtumaan tavallisissa työtilanteissa sijainnista riippumatta. Koska kuvaamisen suorittaa todennäköisimmin kiireinen asentaja tai korjaaja, on kuvaamisen tapahduttava asennus- tai korjaustyön ohessa, sitä häiritsemättä. Laitteiston asentaminen ei saa viivästyä siksi, että operaatio

kuvataan. Kuvaamisen tulee myös voida tapahtua missä vain, milloin vain. Kuvaustyökalun kannalta tämä tarkoittaa, että sen on oltava aina saatavilla ja valmiina kuvaamaan. Kuvaamisen valmistelu ja oheistoiminnot on voitava suorittaa varsinaista työtä merkittävästi hidastamatta, jottei videon laatimisen hinta nouse liian korkeaksi. Onhan tuotteen alkuperäinen ideakin lähtöisin tuottavuudesta.

3.4 Ominaisuudet

Videon kuvaussovellus käyttää matkapuhelinlaitteen sisäänrakennettua kameraa. Kameran on kyettävä riittävän suureen tarkkuuteen, jotta tuloksena syntyvä video olisi katseltavissa myös työaseman näytöllä. Kuvattu video tallennetaan puhelimen muistiin tai puhelimeen asennetulle muistikortille. Tallennetun videon katselemista varten on käytettävä myös videon toistotoimintoa. Sovelluksen perusominaisuutena on myös videon lähetys palvelimelle videopankkiin. Palvelinyhteyden luomiseksi on käytettävä matkapuhelimen verkkoliikenneominaisuuksia. Yhteyteen käytetään vakiintunutta yhteyskäytäntöä, jotta palvelinohjelmisto osaa varmasti ottaa videon vastaan asianmukaisesti. Sovellus käyttää myös GPS-paikannuslaitetta, mikäli sellainen on käytettävissä. GPS-yksikkö voi olla puhelimen sisäinen tai erillinen puhelimeen kytketty lisälaite.

3.5 Laiteriippumattomuus

Kuvausohjelmisto tulisi käyttöön potentiaalisesti hyvin heterogeeniselle joukolle ihmisiä, joten heillä jo käytössään olevien puhelimien kirjokin olisi luultavasti suuri. Uusia puhelimia ei turhaan tahdota ostaa, joten videon kuvaussovellus tulisi voida asentaa mahdollisimman moneen jo olemassa olevaan puhelimeen. Tuote tulisikin pyrkiä tuottamaan niin, että se on käytettävissä mahdollisimman monessa erityyppisessä puhelimessa merkistä tai mallista riippumatta. Laitteistosta riippumattomuus parantaisi mahdollisuuksia tarjota tuotetta mahdollisimman suurelle asiakasjoukolle.

Käytettävien laitteiden joukkoa rajoittaa kuitenkin käytännössä esimerkiksi joidenkin puhelimien tarjoama heikko tallennettavan videon laatu. Myös tallennustilan pieni mää-

rä voisi muodostaa esteen joidenkin laitteiden kohdalla. Käytännössä voikin olla, että mobiilisovellus joudutaan tekemään tietylle alustalle kohdennettuna. Käyttöliittymät ovat erilaisia eri laitteissa, joten käytetyistä testilaitteista ominaisuuksiltaan poikkeavilla puhelimilla sovellusta voisi olla hankala käyttää. Kohdistettaessa sovellus tietylle alustalle saataisiin laitteen erityisominaisuudetkin käyttöön [12, s. 3].

3.6 Kehitystyön sujuvuus

Suuri osa ohjelmistotuotteen tuottamiseen tarvittavasta investoinnista kohdistuu sovelluskehitystyöhön. Kehitystyön tulisikin olla sujuvaa ja johdonmukaista tuotoksen käyttöjärjestelmäalustasta tai ohjelmointikielestä riippumatta. Tuotteen jatkokehityksen seuraavaan versioon on oltava myös jouhevaa. Siten muodostuu tärkeäksi, että nekin ominaisuudet, joita on harkittu ensimmäiseen versioon, mutta ei nähty vielä tarpeellisiksi, voitaisiin toteuttaa valitulla toteutustavalla. Ohjelmistotuotteita valmistavan yrityksen eduksi siis on, jos kehitystyön sujuvuuteen kiinnitetään huomiota tuotteen toiminnallisten ominaisuuksien lisäksi.

Tätä päivää on myös älypuhelinsovellusten testaus- ja hyväksymispalvelujen ostaminen. Jokainen kunnollinen puhelimeen saatava sovellusohjelma tulisi olla hyväksytty. Tämä näyttäisi pitkittävän jonkin verran sovelluskehitysprosessia, mikäli tarvitaan rajoitetusti käyttöön sallittavia järjestelmäpalveluja. Nyt läpikotaisen testauksen saa kuitenkin kätevästi keskitettynä palveluna. Kaupan päälle saa vielä leiman, joka kertoo sovelluksen olevan sopimusten mukainen, turvallinen ja asiallinen. Hyväksytys kannattaa ehdottomasti tehdä valmiille sovelluksille. [13, s. 1–2; 14.]

3.7 Elinkaari

Nyt valmistettavan tuotteen tulisi toimia myös tulevaisuudessa mahdollisimman monella laitteella. Älypuhelinalan tulevaisuuden suuntausten tarkastelu on siis tarpeen myös sovelluksen suunnittelussa. Alalla muutokset voivat olla hyvinkin nopeita, ja ohjelmistotuotteen tuottaneelle yritykselle merkityksellisiä. Vaikeaa yhteensopivuuden varmis-

tamisesta tekee teknologioiden nopea uusiutuminen. On vaikea ennustaa, mitä teknologioita tietty matkapuhelinvalmistaja käyttää muutaman vuoden päästä. Vaikka nyt käytössä olisi lähes yksinomaan esimerkiksi jokin tietty käyttöjärjestelmä, voi tilanne vuoden päästä olla aivan toinen. Eroja siis on jopa saman valmistajan eri tuotteiden välillä, valmistajakohtaisista puhumattakaan. Älypuhelinmaailman teknologioita syntyy ja tuhoutuu tiuhaa tahtia. Se on hyvä asia uudistumisen kannalta, mutta vaikeuttaa tulevan ennustamista sovellusten elinkaarta suunniteltaessa.

Sovelluksen asentaminen tulee olla helppoa. Ylläpitoa se ei saa vaatia lähes lainkaan. Päivitykset ohjelmaan pitää olla helppo ja suoraviivainen asentaa, ja vanhojen asetusten ja tietojen on säilyttävä tallessa myös päivitettäessä. Ohjelman asetukset ja videotiedostot sekä videoiden tiedot on myös liitettävä puhelimen varmuuskopiointitoimiin, jotteivät tärkeät tiedot katoa vaikkapa puhelimen käyttöjärjestelmäpäivityksen yhteydessä.

3.8 Tietoturva

Tallennettaessa salasanoja on käytettävä salausta, jonka on todettu antavan riittävän turvan salasanatiedolle. Käyttäjätietojen lähetys palvelimelle suoritetaan käyttäen salattua HTTPS-protokollaa. Mitään ylimääräisiä tietoja koskien käyttäjän puhelinlaitetta, ei lähetetä palvelimelle. GPS-yksikön käyttö rajoittuu paikkatiedon tallentamiseen videon tietoihin eikä sitä käytetä muutoin.

4 Suunnittelu

4.1 Järjestelmäsuunnittelu

Kun MentorAid-järjestelmän mobiililaajennusta alettiin suunnitella, oli tiedossa ainoastaan, että videot kuvataan matkapuhelimella ja niiden oli lopulta päädyttävä palvelimelle katseltaviksi. Kyseessä oli siis asiakas-palvelinarkkitehtuuriin pohjautuva sovellus. MentorAid-järjestelmään on jo aiemmin toteutettu tiedoston vastaanottomahdollisuus, jota voidaan pienin muutoksin käyttää myös opetusvideokokonaisuuksien vastaanot-

toon. Matkapuhelimen toiminnan osalta pohdittiin myös, voisiko olemassa olevaa kamerasovellusta ja median lähetystoimintoa käyttää myös opetusvideoiden lähetykseen MentorAid-järjestelmään. Käytettävyysvaatimus ei kuitenkaan täyttyisi, koska puhelimen kamerasovelluksen lähetysominaisuudet eivät ole helposti löydettävissä toimintoon perehtymättömälle käyttäjälle. Myöskään lisätietojen lähettäminen videoon liitettynä ei onnistu olemassa olevilla työkaluilla.

MentorAid-palvelin osallistuu videoiden käsittelyyn vasta, kun kuvatut videot lähetetään tallennettaviksi palvelimelle. Tällöin vaaditaan päätelaitteen tunnistamista ennen kuin voidaan muodostaa videon siirtoyhteys. Tämä tunnistaminen suunniteltiin toteutettavaksi niin, että matkapuhelimessa ajettava sovellus pyytää ensin HTTP-istunnon aloittamista palvelimelta. Palvelin kysyy seuraavaksi puhelinsovellukselta käyttäjätunnuksen ja salasanan, ja varmistaa niiden oikeellisuuden. Näiden tietojen varmistuttua oikeiksi palvelin antaa pääsyn osaamisenhallintajärjestelmään ja käynnistää istunnon. Nyt videon ja lisätietomääreiden lähetys voi alkaa. Video lähetetään palvelimelle HTTP Post -pyynnön avulla. Post-pyyntöön liitetään parametreiksi kaikki lähetettävät lisätiedot, jotka palvelin purkaa kutsusta ja tallentaa tietokantaan. Videon tietoihin tallennetaan myös vastaanotetun tiedoston tiedostopolku videon toistoa varten.

Videon toistoa varten on MentorAid-järjestelmään lisättävä mahdollisuus upottaa videosisältöä HTML-sivuun. Sovellusohjelmia upotukseen on paljon, mutta matkapuhelinten yleisintä videon tallennusmuotoa, H.263:a, tukevaa toisto-ohjelmaa ei löydetty. Tässä yhteydessä sopivimmaksi toisto-ohjelmaksi todettiin Flowplayer, sillä se vaikutti ammattimaiselta tuotteelta, jolla on riittävä tuotetuki ongelmatilanteita varten. Flowplayer toistaa videoita, joiden tiedostomuoto on joko Flash-video tai H.264-standardin mukaan pakattu MPEG-4 [15]. Nämä videon toistomuodot ovat riittävät, sillä alustavan tutkimuksen mukaan ainakin H.264-pakatun MPEG-4-videon tallennus onnistuisi uusilla älypuhelimilla [16, s. 3]. Puhelimeen asennettavan sovelluksen on siis tuettava jompaakumpaa näistä tallennusmuodoista.

4.2 Mobiilisovelluksen suunnittelu

4.2.1 Toteutusalustojen vertailu

Matkapuhelimissa on käytössä useita eri käyttöjärjestelmiä ja niiden versioita. Symbian OS -käyttöjärjestelmällä varustettuja puhelimia myydään maailmalla eniten, mutta Applen iPhone OS -käyttöjärjestelmän sisältävät puhelimet yleistyvät kovaa vauhtia [17]. Suomessa, johon laadittavaa ohjelmistotuotetta ollaan pääosin kohdentamassa, Symbian OS pysynee selvästi yleisimpänä älypuhelinten käyttöjärjestelmänä.

Symbian-käyttöjärjestelmän oma ohjelmointikieli Symbian C++ otetaan mukaan mahdollisten toteutuskielien vertailuun. Sen kiistämättöminä etuina ovat ohjelmointirajapintojen runsaus ja sovellusten toimintanopeus. Haittapuolena Symbian C++:ssa on hidas ohjelmointityö ja vaikea kielen oppiminen alustaa tuntemattomalle ohjelmoijalle. Symbian C++ rajoittuu myös vain Symbian OS -puhelimiin.

Symbian C++ -sovellukset pitää hyväksyttää myös kehitysvaiheessa, jos käytetään tiettyjä alustan kyvykkyyksiä (capabilities). Kehitysvaiheen toistuvista hyväksytyksistä voisi aiheutua viivettä kehitysprosessiin. Alustavasti useimmat ominaisuudet voidaan kuitenkin toteuttaa ilman kehityksenaikaista hyväksyttämistä. Tarvittavista ominaisuuksista ainoastaan paikannuksen käyttö näyttäisi vaativan allekirjoituksen myös kehitysvaiheessa. [14.]

Kun pohditaan älypuhelinohjelmiston toteutusta ja käytettävissä olevia sovelluskehitysalustoja, on aina otettava huomioon Java-ohjelmointikielen mobiililaitteiden ohjelmointiin soveltuva Java Micro Edition. Nykypuhelimiin voidaan lähes käyttöjärjestelmästä riippumatta asentaa Java ME -sovelluksia. Kehitystyö on Java ME:llä nopeata, mutta ohjelmointirajapintojen niukkuus voi rajoittaa joidenkin toimintojen toteutusta [12, s. 4].

Kolmantena varteenotettavana toteutuskielenä on mainittava S60-ohjelmistoalustalle tehty versio Python-ohjelmointikielestä, Python for S60 (PyS60). Pythonilla ohjelmointityö on nopeata. Haittapuolena on Symbian C++:sta tuttu laiteriippuvuus. Pythonia käytetään ohjelmistonkehitysnopeutensa ansiosta yleisesti suuritöisten sovellustuotteiden alkuvaiheen prototyyppien tekoon.

Vaatimusanalyysin pohjalta lähdettiin vertailemaan näitä kolmea ohjelmointikieltä. Koska Pythonin ohjelmistokehityksen nopeudesta ei ollut edes luotettavaa arviota, päätettiin sovelluskehityksen nopeutta arvioida käytännössä. Metropolia Ammattikorkeakoulussa tehdyn harjoitustyön tuloksena syntynyttä Symbian C++ -sovellusta käytettiin ohjelmistokehityksen nopeuden viitearvona kielelle. Kyseinen sovellus kirjoitettiin, kun kieltä oli opiskeltu noin kuukausi, ja se haki Internetistä XML-tiedoston sekä etsi siitä määrättyjä tietoja. Symbian C++:n kehitysnopeuden viitearvoksi saatiin noin kaksi työpäivää, eli 16 tuntia. Samansisältöinen sovellus tehtiin myös PyS60:lla (ohjelmakoodilistaus liitteessä 1). Sovellus valmistui yhdessä työpäivässä sisältäen kielen opiskelun. Sam Mason ja Elise Korolev arvioivat Java ME -sovelluskehityksen olevan 80 prosenttia C++ -kielistä kehitystyötä nopeampaa [12, s. 3]. Java ME:llä esimerkkisovelluksen tekoon olisi siis mennyt tuon arvion mukaan noin 3,2 tuntia. Kun Python-kehityksen yhden työpäivän tuloksesta vähennetään työkalujen asentamiseen ja kielen opiskeluun kulunut aika, jää ohjelmoinnin osuudeksi arviolta 4 tuntia. Tästä voitiin tehdä karkean luokan johtopäätös, että Java ME sekä PyS60 ovat sovelluskehitystyön nopeudessa hyvin lähellä toisiaan. [18.]

Kartoitettaessa ohjelmointikielten ominaisuuksia päädyttiin seuraaviin tiivistettyihin listoihin eri vaihtoehtojen hyvistä ja huonoista puolista.

Java Micro Edition

Edut

Ei ole sidoksissa yhteen käyttöjärjestelmäalustaan. Alustakohtaista vaihtelua ominaisuuksissa on nähtävissä. [12, s. 4–5, 8.]

Saman sovelluksen pitäisi toimia tulevaisuuden käyttöjärjestelmäversioissa.

Kehitystyö on nopeata, noin 80 % nopeampaa kuin C++:lla [12, s. 3].

Java on jo ennestään käytössä yrityksessä.

Haitat

Järjestelmän ominaisuuksia on käytettävissä rajoitetusti [12, s. 4].

Ohjelmatiedostoja ei voi päivittää, vaan on asennettava uusi versio vanhan päälle. Joitakin asetuksia voi tuoda vanhasta versiosta uuteen. [12, s. 4.]

Sovellus käynnistyy hitaammin kuin C++:lla tehty [12, s. 4].

Ei tue tarvittavien videomuotojen tallennusta.

Symbian C++

Edut

Kaikki järjestelmän ominaisuudet ovat käytössä [12, s. 4].

Saman sovelluksen pitäisi toimia tulevaisuuden käyttöjärjestelmäversioissa.

Hyvät jatkokehitysmahdollisuudet, koska kaikki järjestelmäpalvelut ovat käytössä.

Ohjelmatiedostot voi päivittää helposti päivityspaketeilla [12, s. 4].

Sovellus käynnistyy ja toimii nopeasti [12, s. 2–4].

Haitat

Käyttöjärjestelmäriippuvainen

Kehitystyö on hidasta [12, s. 3].

Pakollinen Symbian Signed -hyväksytys saattaa hieman hidastaa kehitysprosessia.

Python for S60 (PyS60)

Edut

Kehitystyö on huomattavan nopeata.

Haitat

Laiteriippuvainen kuten C++

Ei kunnon dokumentaatiota eikä tuotetukea

Pakollinen Symbian Signed -hyväksytys saattaa hieman hidastaa kehitysprosessia.

4.2.2 Toteutusalustan valinta

Java Micro Edition oli hyvin vahvana ehdokkaana tuotteen toteutuskieleksi, sillä se oli erinomainen valinta miltei kaikilla mittareilla. Java ME:n kanssa tasavahvana oli Symbian C++, joka tarjosi laajemman valikoiman ohjelmointirajapintoja laitteen erityisominaisuuksien käyttöön. Tarkasteltaessa Java ME:n videon tallennusmuotojen valikoimaa ei kuitenkaan kyetty löytämään tietoa siitä, pystyykö Java ME:llä tallentamaan videota H.264-pakatussa MPEG-4-muodossa. Tästä syystä Java ME jouduttiin hylkäämään ohjelmointikielimahdollisuutena.

Jäljelle jääneistä toteutusvaihtoehdoista Pythonilla on nopea ohjelmoida, mutta epäluotettavaksi sen tekee kunnollisen dokumentaation ja tuotetuen puute. Tuotetukea ja dokumentaatiota sitä vastoin löytyy Symbian C++:sta ja S60-ohjelmistoalustasta hyvin. Symbian C++ -kielestä oli kehitysryhmässä lisäksi jo hieman kokemusta, joten tuotteen toteutuskieleksi valittiin Symbian C++. Tärkeimpinä kriteereinä valinnalle olivat tuotetuki, videon tallennusformaattien riittävyys sekä aiempi kokemus kehitystyöstä C++:lla.

Lähinnä Suomessa käyttöön tarkoitetulle sovellukselle toteutus Symbian C++:lla tarkoitti käytännössä S60-alustalle ohjelmoimista sen laajaan levinneisyyteen perustuen. Symbian-käyttöjärjestelmässä on yhteensopivuuskatko ennen versiota 9, joka on käytössä S60-alustan kolmannessa painoksessa (S60 3rd Edition). [19]. Vanhempien versi-

oiden konekieliset ohjelmat eivät siis toimi uudemmissa versioissa. Tämän lisäksi ohjelmakoodi ei käänny ongelmitta vanhasta versiosta uudelle. Tulevaisuuden yhteensopivuuden varmistamiseksi päätettiin ohjelmoida tuote kohdennettuna yhteensopivuuskatkon jälkeen julkaistuille versioille. Käyttöön otettiin siis S60 3rd Edition.

4.2.3 Käyttöliittymä

Käyttöliittymäsuunnittelussa ensimmäinen askel oli määritellä mahdolliset näkymät ja se, kuinka ne vaihtuvat eri tilanteissa. Näkymät hahmoteltiin kynän ja paperin avulla, jotta saatiin piirrokset aikaan nopeasti. Piirtämisen nopeus vähentää piirtämisen aikaista itsekritiikkiä ja pitää näkymien ulkonäön suunnittelun irrallaan teknisistä rajoituksista. Kaikkien suunniteltujen näkymien hahmotelmat ovat liitteessä 2. Näkymien lukumäärä tuli pitää alhaalla, jotta käytettävyys saatiin maksimoitua. Aluksi kaavailtiin oletusnäkymäksi päävalikkoa, josta olisi mahdollisuus siirtyä johonkin varsinaiseen toimintonäkymään; videolistaan, videon katseluun, videokuvaukseen tai asetuksiin. Kuvassa 1 näkyvä päävalikkokonsepti kuitenkin hylättiin, koska käytettävyys kärsisi liian monesta askeleesta siirryttäessä sovelluksen varsinaiseen käyttönäkymään. Päävalikko ei myöskään toisi merkittävää parannusta käyttöliittymän selkeyteen.

Kuva 1. Pois jätetty päävalikkonäkymä

Päänäkymäksi valittiin lista jo kuvatuista videoista. Muita näkymiä ovat asetus- kuvaus- ja katselunäkymät. Päänäkymän listassa näkyy videon nimi ja kuvaus. Listasta videon valittuaan käyttäjä pääsee muokkaamaan videon nimeä, kuvausta sekä lisätietoja. Listanäkymästä voi valikon kautta siirtyä joko kuvaamaan uutta videota tai jo kuvatun videon uudelleenkuvaukseen. Uudelleenkuvaus korvaa vanhan videon ja ottaa sen lisätiedot. Päänäkymästä voi myös siirtyä katsomaan jo kuvattuja videoita.

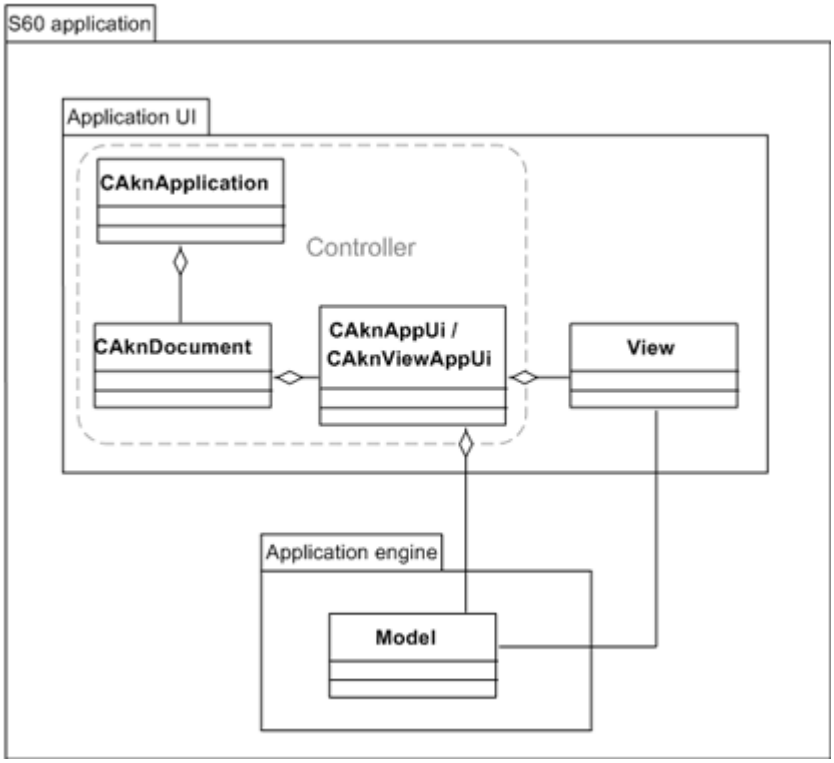
Tallennusnäkymässä näytöllä on vain kameran etsin kohteen näyttämiseen, ja tallennuksen voi aloittaa valikosta tai keskimmäisellä valintanäppäimellä. Tallennuksen lopetus tapahtuu myös keskinapista. Tallennettaessa videota näkyy ruudulla tallennusilmaisin,

jotta käyttäjä tietää mitä tapahtuu. Tallennusilmaisimena voi toimia esimerkiksi kuvake tai lyhyt teksti. Katselunäkymä on ulkonäöltään täysin samanlainen kuin tallennusnäkymäkin. Ruudulla näkyy videon ensimmäinen kuvaruutu ja käyttäjä voi käynnistää videon toiston valikosta tai keskimmäisestä valintapainikkeesta. Asetusnäkymässä näkyy lista muutettavissa olevista asetuksista, esimerkiksi vastaanottavan palvelimen osoite ja videotiedostojen tallennuspaikka.

Näkymien käyttöliittymäsuunnittelussa oli tärkeää pitää yksittäisen näkymän elementtien määrä pienenä, jotta näkymä pysyy selkeänä ja helppona hahmottaa. Videon katselunäkymään ideoitiin myös tässä vaiheessa uutena ominaisuutena yksinkertainen videon muokkaus. Katsottaessa videota matkapuhelimella voitaisiin videoon merkitä toiston aloitus- ja lopetuskohtia, joita videota toistettaessa käytettäisiin ei-toivottujen videon osien jättämiseen toiston ulkopuolelle. Videota toistavan palvelinpään sovelluksen pitäisi tietysti osata myös tulkita näitä toistonohjausmerkkejä.

4.2.4 Arkkitehtuuri

Ohjelmiston arkkitehtuurin suunnittelussa tärkeätä oli, että sen tulee noudattaa Symbian-käyttöjärjestelmän käytännön mukaisesti kolmikerroksista MVC-arkkitehtuuria erottelemalla tietorakenteet (model), näkymän (view) ja käyttöliittymän (controller) toisistaan [20]. Tietorakenteiden, näkymien ja käyttöliittymän välinen yhteys on rajoitettu näiden välisiin funktiokutsuihin, joilla tietoa haetaan tietorakenteista tai muutetaan näytettäviä objekteja näytöllä. Kuvassa 2 nähdään sovelluksen yleinen rakenne jaoteltuna MVC-arkkitehtuurin mukaan.



Kuva 2. S60-sovelluksen MVC-arkkitehtuurin mukainen rakenne [20].

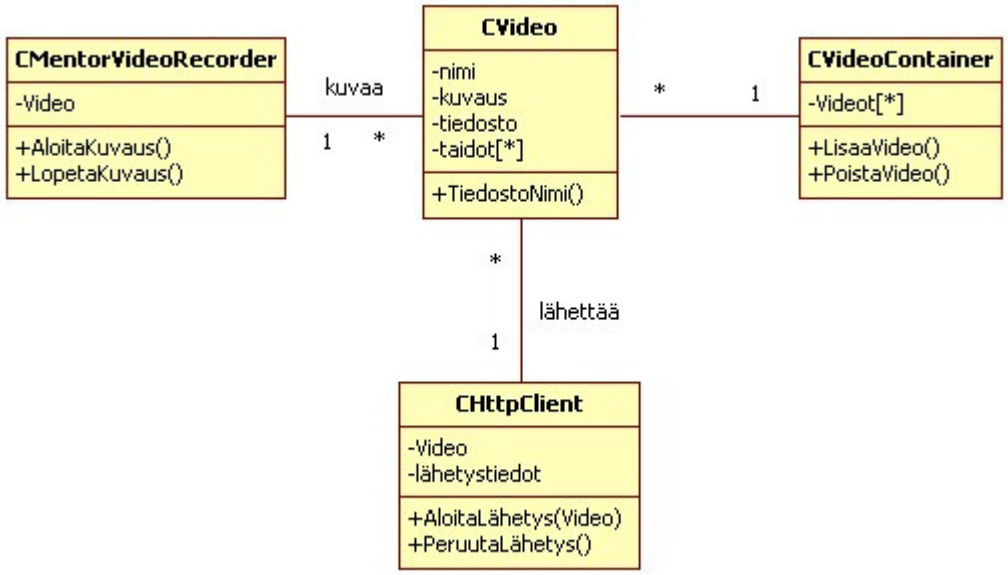
Koska sovelluksessa tulisi olemaan useita selkeästi toisistaan erotettavia näkymiä, kukin eri toiminnoin varustettuna, päätettiin sovelluksessa ottaa käyttöön S60-alustan näkymänvaihtoarkkitehtuuri. Näkymänvaihtoarkkitehtuurin käyttö mahdollistaa näkymien vaihtamisen lisäksi eri toiminnallisuuksien erottelun lähdekoodissa, mikä parantaa ohjelmakoodin selkeyttä. Muistinhallinnan näkökulmasta saadaan lisäksi vähennettyä resurssien turhaa varaamista, koska näytöstä poistuneen näkymän kontrollit ja muut tilapäiset tiedot voidaan poistaa muistista seuraavaan aktivoitumiseen saakka.

Ohjelmiston on tallennettava tilansa pysyväismuistiin ohjelman sulkemisen yhteydessä. Tätä tarkoitusta varten on otettava käyttöön Symbian OS -alustan sovelluskehyksen tarjoamat tiedon pysyvyyspalvelut. Sovelluksen lopetuksen yhteydessä tallennettavaksi tiedoksi luokiteltiin videoiden osalta videoiden nimet, niiden kuvaukset sekä kaikki lisätiedot. Sovelluksen asetukset oli myös tallennettava. Video tallennetaan kuvattaessa suoraan pysyväismuistiin videotiedostoon, joten tiedostonimen talletus videon tietoihin riittää tässä yhteydessä.

Videon kuvaukseen käytetään Symbian-käyttöjärjestelmän multimediakehysrakennetta, jonka tarjoamilla palveluilla video saadaan kuvattua sekä toistettua. Videon tallennusmuodon valinta tapahtuu myös tämän kehysrakenteen tarjoamien palveluiden avulla. Video tallennetaan kuvauksen yhteydessä pysyväismuistiin käyttäjän valitsemaan paikkaan, joko puhelimen muistiin tai muistikortille. Tallennettaessa video pakataan H.264-standardin mukaan, jotta videon toistosovellus pystyy sitä suoraan toistamaan. [9, s. 58.]

Myös videon lähetykseen käytetään Symbian-käyttöjärjestelmään rakennettua valmista kehysrakennetta. Lähetystä varten tarvitaan HTTP-kehysrakenteen tarjoamia palveluita.

Sovelluksen tietorakenteet pyrittiin pitämään yksinkertaisina ja tehokkaina turhan resurssien varaamisen välttämiseksi. Perusosaksi suunniteltiin yksinkertainen CVideo-luokka, joka edustaa videota ja sisältää sen tiedot. Videoille annetaan järjestysnumerot luontijärjestyksessä yksilöinnin helpottamiseksi. Lisäksi on toteutettava tiedon pysyvyystoimintojen vaatimat funktiot CVideo-luokalle, jotta tarvittavat tiedot saadaan tallennettua ja palautettua. Tietojen pysyvyyspalveluita tarvitaan sovelluksen sulkemisen ja avaamisen yhteydessä. CVideo-oliot talletetaan CVideoContainer-säiliöön, joka on mahdollista tarjota suoraan valintalistalle (list box) sisällöksi. Tälle luokalle on myös toteutettava tiedon pysyvyyteen vaadittavat funktiot. Tiedonsiirtoyhteyksiä varten on listanäkymän alaisuuteen sijoitettava HTTP-asiakasluokka, CHttpClient, joka hoitaa yhteyksien muodostuksen, käytön ja sulkemisen. Listanäkymä on tälle luokalle looginen paikka, koska listalta valitaan lähetettävä video ja annetaan lähetyskomento. Kuvaamiseen tarvittavat apuluokat sijoitetaan kuvausnäkymän alle. Kuvassa 3 näkyvällä luokkakaaviolla hahmoteltiin sovelluksen tietorakennetta suunnitteluvaiheessa.



Kuva 3. Sovelluksen tietorakenteen suunnittelunaikainen luokkakaavio.

Lokalisoituja kieliversioita varten otetaan käyttöön kaikkien käyttäjälle näytettävien tekstien luku resurssitiedostoista. Tällä menetelmällä saadaan uudet kieliversiot lisättyä tuotteeseen kääntämättä ohjelmakoodia uudelleen. Menetelmä sallii myös helpon jaottelun eri tiedostoihin kieliversion perusteella, joten erikielisiä tekstiresursseja on helppo hallinnoida.

4.3 Toteutusprosessin suunnittelu

Toteutusprosessin oli oltava selkeästi jaettavissa eri vaiheisiin työn seurannan sekä sovelluksen modulaarisen suunnitelman seuraamisen helpottamiseksi. Ohjelmointityö on lisäksi selkeää ja johdonmukaista toteutettaessa ohjelmointi hyvin rajatuissa osissa, koska tällöin voidaan keskittyä yhteen aihealueeseen kerrallaan. Jotta vältyttäisiin ohjelmiston osien uudelleenohjelmoinnilta eri kehityksen vaiheissa, katsottiin järkeväksi luoda heti tietorakenteiden jälkeen käyttöliittymärunko, johon sovelluksen muut ominaisuudet voitaisiin suoraan upottaa jo valmistusvaiheessa. Osien testaus valmiin käyttöliittymän avulla säästää aikaa erillisten testiohjelmien käyttöön verrattuna. Sovellusta

olisi myös helppo esitellä vajavaisellakin toiminnallisuudella, jos näin haluttaisiin. Päätettiin, että ensimmäisestä prototyypistä jätetään pois asetusnäkymä sekä GPS-paikannus.

Yksikkötason testaus kuuluu olennaisena osana toteutusprosessin jokaiseen vaiheeseen. Sovelluksen osaa ohjelmoitaessa sitä testataan jatkuvasti. Valmistuessaan se testataan vielä osana kokonaisuutta. Ohjelmointityön aikainen testaus ei kuitenkaan yksin riitä, vaan tarvitaan mittavampaa testausta, kun koko sovellusohjelma on koottu yhteen.

5 Toteutus

5.1 Työkalut

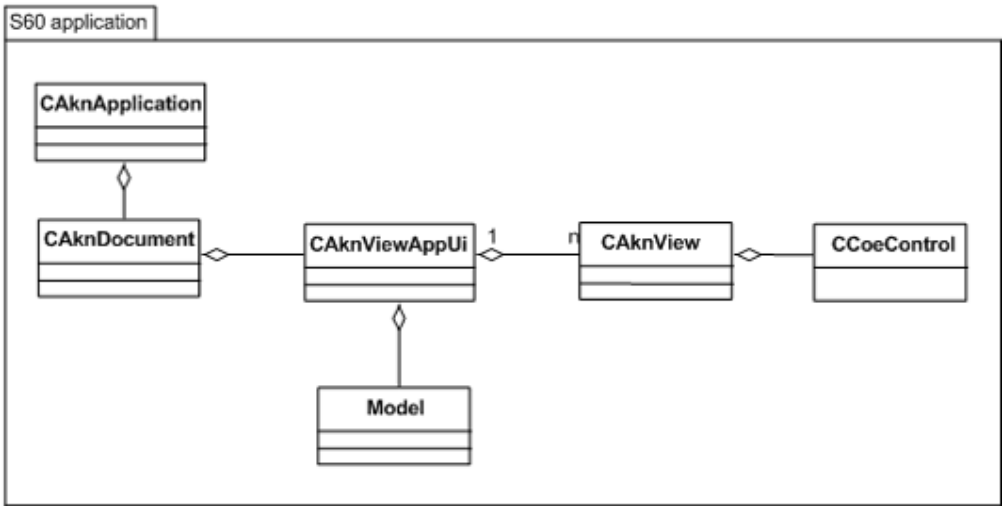
Toteutusvaiheen ohjelmointityössä on käytetty erilaisia työkaluohjelmia. Käytössä on ollut Carbide.c++ 2.0 -kehitysympäristön sovelluskehittäjille suunnattu Developer-versio, joka sisältää hyvät työkalut sovellustuotantoon S60-mobiililaitteille. Tarjolla on hyvät ohjelmakoodin täydennys- ja tarkistusominaisuudet sekä välineet emulaattorissa ja myös suoraan kohdelaitteessa tapahtuvaan virheenetsintään. Käytössä on ollut myös Nokia N95 -matkapuhelin sekä sen käyttöjärjestelmäversiota vastaavat, S60 3rd Edition Feature Pack 1 -SDK:n (software development kit) ohjelmistonkehitystyökalut. Apuna on käytetty myös esimerkiksi Notepad++ -ohjelmaa esimerkkikoodin tarkasteluun ja palvelinpään videon toistokoodin hahmotteluun. Älypuhelinohjelmistoa on myös testattu N95-puhelimessa laitteelle asennettavan Application TRK -virheenetsintätyökalun avulla. HTTP-yhteyden ohjelmoinnissa käytettiin apuna Ethereal-ohjelmaa verkkoliikenteen seurantaan. Palvelinympäristön ohjelmakoodin kirjoituksessa on käytetty MentorAid-projektin yhteydessä yleisesti käytettyä kehitysympäristöä, johon kuuluu esimerkiksi Java-ohjelmointikieli, Spring ja Hibernate kehysrakenteet, Eclipse IDE -sovelluskehitysympäristö, MySQL-tietokanta, sekä Apache Tomcat -Web-palvelin. Ohjelmakoodin versionhallintaan on käytetty CVS-palvelinta.

5.2 Järjestelmän kuvaus

Mobiili videontuotantojärjestelmä jaettiin toteutusta varten viiteen erilliseen osaan. Osakokonaisuudet olivat käyttöliittymä, tietorakenteet, tietoliikenne, videon kuvaus sekä videon vastaanottamiseksi palvelinohjelmistoon tehtävät muutokset. Osat ohjelmoitiin erillisinä kokonaisuuksina ja testattiin samalla yksikkötasolla sekä jo valmistuneiden komponenttien yhteydessä.

Toteutus aloitettiin tietorakenteiden ohjelmoinnilla, jotta toteutuksen muilla osa-alueilla tarvittavat reaalimaailman asiat saatiin mallinnettua. Tietorakenteiden ohjelmointi käsitti määriteltyä videokokonaisuutta mallintavan CVideo-luokan sekä video-olioiden varastointiin ja hallintaan tarkoitetun CVideoContainer-luokan toteutukset. Luokille oli toteutettava tiettyjä tiedon pysyvyys- ja tallennustoimintoja, jotta Symbian-käyttöjärjestelmän sovellusrakenne voi niiden ilmentymät tallentaa automaattisesti.

Rakennettaessa sovellusta, jossa käytettävyydelle asetetaan korkeita vaatimuksia, on käyttöliittymän toiminta ja sen toteutuksessa käytettävät tekniikat harkittava huolella. Käyttöliittymän toteutuksessa käytettiin S60-näkymänvaihtoarkkitehtuuria, joten sen määrittämät ohjelmiston rakenteen vaatimukset on otettu myös huomioon. Näkymänvaihtoarkkitehtuurin käyttöön tarvittavien luokkien luokkarakenne näkyy kuvassa 4. Käyttöliittymän toteuttavat luokat on siis peritty kuvassa näkyvistä S60-alustalle toteutetuista kantaluokista.



Kuva 4. S60-näkymänvaihtoarkkitehtuurin luokat [21].

Sovelluksen ydintoiminto, videon kuvaus, tehtiin käyttäen Symbian-multimediakehyksen palveluita. Kameran hallintaa varten kirjoitettiin CCameraEngine-luokka, jonka vastuulla on varata kameralaite ja näyttää kameran etsimen kuva näytöllä. Videon kuvaukseen tehtiin erillinen CMentorVideoRecorder-luokka, joka määrittelee videokuvauksen asetukset sekä kuvaa videon tallentaen sen pysyväismuistiin.

Tietoliikenneyhteys toteutettiin käyttäen HTTP-kehysrakenteen palveluita. HTTP-liikenne toimii asynkronisesti eikä siten häiritse puhelimen muita toimintoja. Videon palvelimelle lähettävä CHttpClient-luokka toimii HTTP-asiakkaana siirtäessään tietoa palvelimelle.

Palvelinpäähän videopakettia vastaanottamaan muokattiin olemassa olevasta tiedoston vastaanottotoiminnosta määriteltyjen videoiden vastaanottamiseen soveltuva versio. Opetusvideon näyttämistä varten toteutettiin myös palvelimen tarjoaman JSP-sivun (Java server pages) sisältöön upotettava videontoistotoiminto.

5.3 Tietorakenteet

5.3.1 Yleisperiaatteet

Symbian käyttöjärjestelmässä sovellusten toiminta pohjautuu sovelluskehyksen tarjoamiin palveluihin. Sovelluskehyksen tarjoamat palvelut, kuten tiedon pysyvyys, otetaan käyttöön toteuttamalla itse toteutetuissa luokissa joukko funktioita, joita sovelluskehys käyttää tarvittaessa. Tämän lisäksi on hyvä noudattaa yleisesti käytössä olevaa sovelluksen rakennetta, jotta ohjelmakoodi rakentuu loogisesti ja selkeästi. Huomionarvoista on, että `CVideoContainer`-malli on sijoitettu tässä sovelluksessa Document-olion omistukseen, eikä AppUi-luokan omistukseen, kuten kuvassa 2. Muutos oli tehtävä, koska sovelluskehys tukee sovelluksen tietojen tallennusta Document-luokan kautta [22, s. 9].

Kaikki sovelluksenlaajuisesti käytössä olevat yksittäiset vakiot ja luetellut vakiot (enumeration) määritellään MentorRec.hrh-tiedostossa. Kyseinen tiedosto voidaan sisällyttää sekä resurssitiedostoon että lähdekooditiedostoon. MentorRec.rss-resurssitiedostossa

määritellään kaikki vakiomerkkijonot, ja ne ovat siten käytettävissä kaikkialla ohjelmakoodissa. Resurssitiedostossa määritellään myös valikot, näkymät ja videon tietojen syöttämislomake.

Application-oliolla tulee olla pääsy Document-olioon tiedon pysyvyyspalveluita käytettäessä. Model voidaan lisäksi luoda Documentin alle. **CMentorRecDocument**-luokkaan on tiedon pysyvyyden toteuttamiseksi myös kirjoitettava joitakin funktioita sovelluskehyksen kutsuttaviksi. Sovelluskehys käyttää funktioita kutsuttaessa **SaveL**-funktiota **CMentorRecAppUi**-oliosta juuri ennen sovelluksen sulkemista.

CMentorRecDocument-luokalle on kirjoitettava **OpenFileL**-funktion toteutus. Tämä funktio avaa tiedoston dokumenttiolion tilan palauttamista varten parametrina annetulla tiedostonimellä. **OpenFileL**-funktiossa kutsutaan vain kantaluokan vastaavaa funktiota välittäen saadun parametrit suoraan sille. [22, s. 12.]

```
CFileStore* CMentorRecDocument::OpenFileL(TBool aDoOpen,
                                          const TDesC& aFilename,
     RFs& aFs)
    {
    return CEikDocument::OpenFileL(aDoOpen, aFilename, aFs);
    }
```

Kun dokumenttioliolle on avattu tiedosto, käynnistää sovelluskehys dokumentin palautuksen kutsumalla **RestoreL**-funktiota. Funktiossa etsitään tallennettuja pysyvyystietoja sovelluksen yksilöllisellä id-numerolla (UID). Kutsutaan videosäiliön palautusfunktiota, parametreina tallennuspaikka ja sen tietovirran id-numero, jossa ladattavat tiedot sijaitsevat.

```
void CMentorRecDocument::RestoreL(const CStreamStore& aStore,
                                  const CStreamDictionary& aStreamDic)
    {
    TStreamId modelStreamId = aStreamDic.At(Application()->AppDllUid());
    iVideoContainer->RestoreL(aStore, modelStreamId);
    ((CMentorRecAppUi*)iAppUi)->RefreshListViewL();
    }
```

StoreL-funktio on myös toteutettava sovelluskehyksen kutsuttavaksi, kun sovellusta suljettaessa kutsutaan **CMentorRecAppUi**:ssa **SaveL**-funktiota. Funktiossa kutsutaan

videosäiliön tallennusfunktiota **`StoreL`**. Samalla otetaan talteen sen paluuarvona saatu id-numero, joka kertoo mihin tietovirtaan tiedot on tallennettu. Parametrina saatuun tietovirtakirjastoon talletetaan vielä tietovirran id yksilöitynä sovelluksen UID:llä.

```
void CMentorRecDocument::StoreL(CStreamStore& aStore,
                                CStreamDictionary& aStreamDic) const
    {
    TStreamId modelStreamId = iVideoContainer->StoreL(aStore);
    aStreamDic.AssignL(Application()->AppDllUid(), modelStreamId);
    }
```

5.3.2 Video

Videota mallintamaan oli suunniteltu **`CVideo`**-luokka. Videon perustiedot tuli saada asetettua video-olion jäsenmuuttujiin. Lisäksi yksilöintiä helpottamaan asetetaan video-luokan staattisen jäsenmuuttujan avulla videoille id-numerot, jotka alkavat luvusta 1.

```
CVideo::CVideo()
: iId(iNextId++)
    {
    }
```

Video-oliota konstruoitaessa luodaan myös videon tiedostonimi tiedostonimimallin ja id-numeron yhdistelmänä. Tiedostonimi on muotoa "MRV[id].mp4" - eli MRV1.mp4, MRV2.mp4 ja niin edelleen. Tiedostonimen kirjainyhdistelmä on lyhenne sanoista MentorRecorder Video, sovelluksen työnimen mukaan. Videotiedoston nimi ei tule käyttäjälle näkyviin. Tiedostonimi luodaan seuraavasti.

```
RBuf idTxt;
CleanupClosePushL(idTxt);
idTxt.CreateL(KMaxVideoIdLength);
idTxt.AppendNum(iId);
iFileName.CreateL(KFileNameFormat().Length() + idTxt.Length());
iFileName.Format(KFileNameFormat, iId);
CleanupStack::PopAndDestroy(&idTxt);
```

Luotaessa video-oliota annetaan sille myös väliaikainen nimi, joka haetaan vakiomerkkijonona resurssitiedostosta. Käyttäjä voi muuttaa nimen myöhemmin.

```
HBufC* text = StringLoader::LoadLC(R_DEFAULT_VIDEONAME);
SetNameL(*text);
CleanupStack::PopAndDestroy(text);
```

Toteutusvaiheessa huomattiin, että **CVideo**-olioilla olisi hyvä olla funktio, jolla oliosta voisi tehdä kaksoiskappaleen, jotta vältyttäisiin käsin kopioimiselta. Kirjoitettiin siis yksinkertainen **CopyL**-funktio, joka luo uuden **CVideo**-olion ja kopioi alkuperäisen videon tiedot siihen.

```
CVideo* CVideo::CopyL() const
    {
    CVideo* newVideo = CVideo::NewLC();
    newVideo->SetNameL(iName);
    newVideo->SetFormattedNameL(iFormattedName);
    newVideo->SetNameFormatL(iNameFormat);
    newVideo->SetDescriptionL(iDescription);
    newVideo->SetTagsL(iTags);
    CleanupStack::Pop(newVideo);
    return newVideo;
    }
```

Video-olioiden käyttämistä valintalistassa päätettiin helpottaa lisäämällä **CVideo**-luokkaan jäsenmuuttuja **iNameFormat**, johon sijoitetaan listaa luotaessa listan vaatima tekstin muotoilu. Toinenkin jäsenmuuttuja lisättiin tarkoitusta varten, **iFormattedName**, johon sijoitetaan videon muotoillut tiedot. Muotoilun muutostilanteita varten tarvittiin **FormatNameL**-funktio, joka muotoilee tarvittavat tiedot **iFormattedName**-muuttujaan.

```
void CVideo::FormatNameL()
    {
    iFormattedName.Close();
    iFormattedName.CreateL(iNameFormat.Length() + iName.Length() +
    iDescription.Length());
    if (iNameFormat == KDoubleStyleListBoxTextFormat())
        iFormattedName.Format(iNameFormat, &iName, &iDescription);
    }
```

Osoittautui kuitenkin, että tekstin muotoilu valintalistassa käyttöä varten on melko vaikea toteuttaa yleisesti. **Format**-funktioon tarvittavien parametrien määrää ei nimittäin tiedetä, jos valintalistan muoto ei ole tiedossa. Tästä johtuen on käytettävä yllä näkyvän kaltaista ehtorakennetta kaikkien mahdollisten valintalistamuotoilujen oikean toiminnan

varmistamiseksi. Useita muotoiluja haluttiin tukea videoluokassa, jotta vältyttäisiin luokan uudelleenkirjoitukselta valintalistan tyypin mahdollisesti muuttuessa.

CVideo-luokkaan oli myös tiedon pysyvyyden toteuttamiseksi kirjoitettava tiedon kirjoitusfunktio **ExternalizeL**, jonka tehtävänä on kirjoittaa **aOut**-tietovirtaan videon tiedot. Tietovirtaan kirjoitetaan lisäksi muotoilumerkkijonon pituus, jotta tiedon lukemisvaiheessa tiedetään kuinka suuri tila sille on varattava.

```
aOut << iName;
aOut << iDescription;

aOut.WriteInt8L(iNameFormat.Length());
aOut << iNameFormat;
```

Lukuvaiheessa haetaan **InternalizeL**-funktiossa videon tiedot tietovirrasta **aIn**.

```
iName.Close();
iName.CreateL(aIn, KMaxNameLength);
iDescription.Close();
iDescription.CreateL(aIn, KMaxDescriptionLength);
```

Muotoilutieto luetaan ja lukeminen varmennetaan. Muotoilutieto korvataan vakiomuotoilulla, jos sitä ei saatu luettua onnistuneesti.

```
TInt nameFormatLength = aIn.ReadInt8L();
if (nameFormatLength > 0)
    {
    iNameFormat.Close();
    iNameFormat.CreateL(aIn, nameFormatLength);
    }
else
    SetNameFormatL(KDoubleStyleListBoxTextFormat);
```

Lopuksi suoritetaan nimen muotoilu.

```
FormatNameL();
```

5.3.3 Videosäiliö

Videosäiliön tarkoituksena on koota kaikki sovelluksen videot yhteen. Kaikkia videoita voi säiliön kautta käsitellä helposti yhtenä kokonaisuutena. Yksi tärkeä käyttökohde videosäiliölle on sisältötekstien tarjoaminen valintalistalle näytöllä esittämistä varten. Videosäiliöluokka periikin **`MDesCArray`**-rajapintaluokan voidakseen toimia valintalistan tietorakenteena. **`MDesCArray`**-rajapinnan toteuttamiseksi oli videosäiliöluokkaan kirjoitettava alkiot laskeva **`MdcaCount`**-funktio.

```
TInt CVideoContainer::MdcaCount() const
    {
    return iVideos.Count();
    }
```

Toinen **`MDesCArray`**–rajapinnan vaatima funktio on alkion indeksillä osoitus, **`Mdca-Point`**.

```
TPtrC CVideoContainer::MdcaPoint(TInt aIndex) const
    {
    CVideo* video = iVideos[aIndex];
    return video->FormattedName();
    }
```

Yllä voi nähdä **`MdcaPoint`**-funktion palauttavan videon muotoillun nimen, jonka valintalista tämän funktion avulla pyytää. Valintalistan vaatima muotoilu pitää kuitenkin tallentaa säiliön jäsenmuuttujaan video-olioille välittämistä varten. Muotoilu asetetaan videon tietoihin lisättäessä se videosäiliöön.

```
void CVideoContainer::AddVideoL(CVideo* aVideo)
    {
    aVideo->SetNameFormatL(iMdcaTextFormat);
    iVideos.AppendL(aVideo->CopyL());
    }
```

Sovelluksen tietojen pysyvyyden kannalta **`CVideoContainer`**-luokka on keskeisessä roolissa. Luokan tulee olla **`CMentorRecDocument`**-olion omistuksessa ja siihen pitää kirjoittaa tiedon talletus- ja lukutoiminnot, **`StoreL`** ja **`RestoreL`**. Funktioita käyttää dokumenttiolio tiedon pysyvyyskehyksen pyytäessä siltä tietojen tallennusta. **`StoreL`**-

funktiossa tallennetaan tiedot annettuun tallennuspaikkaan (store). Paluuarvona annetaan tallennetun tietovirran id-numero.

```
TStreamId CVideoContainer::StoreL(CStreamStore& aStore) const
    {
    RStoreWriteStream ostream;
    TStreamId id = ostream.CreateLC(aStore);
    ostream << *this;
    CleanupStack::PopAndDestroy(&ostream);
    return id;
    }
```

Vastaavasti **RestoreL**:ssä luetaan annetusta tallennuspaikasta tietovirtaa annetulla id-numerolla.

```
void CVideoContainer::RestoreL(const CStreamStore& aStore,
                               TStreamId aStreamId)
    {
    RStoreReadStream istream;
    istream.OpenLC(aStore, aStreamId);
    istream >> *this;
    CleanupStack::PopAndDestroy();
    }
```

StoreL-funktiossa käytettäessä operaattoria **<<** , kutsutaan automaattisesti videosäiliön funktiota **ExternalizeL**.

```
void CVideoContainer::ExternalizeL(RWriteStream& aOut) const
    {
    TInt16 n = iVideos.Count();
    aOut.WriteInt8L(n);

    for (TInt i = 0 ; i < n ; i++)
        {
        aOut << *(iVideos[i]);
        }
    aOut.CommitL();
    }
```

Samoin kutsutaan **InternalizeL**-funktiota, kun käytetään **RestoreL**-funktiossa **>>** -operaattoria.

```
void CVideoContainer::InternalizeL(RReadStream& aIn)
    {
    iVideos.ResetAndDestroy();
    TInt n = aIn.ReadInt8L();
    for (TInt i = 0 ; i < n ; i++)
        {
        CVideo* c = CVideo::NewL();
        iVideos.AppendL(c);
        aIn >> *c;
        }
    }
```

Videosäiliön asettaminen valintalistan tietorakenteeksi tapahtuu **CMentorRecListContainer**-luokan konstruktorissa. Ensin asetetaan videosäiliön tekstin muotoilu.

```
RBuf textFmt;
CleanupClosePushL(textFmt);
textFmt.Assign(KDoubleStyleListBoxTextFormat().AllocL());
aVideoContainer->SetMdcaTextFormat(textFmt);
CleanupStack::PopAndDestroy(&textFmt);
```

Valintalistan tekstisisällöksi asetetaan videosäiliö. Tekstisisältö saadaan **CVideoContainer**-luokan funktiolla **MdcaPoint**, jota valintalistaolio kutsuu automaattisesti.

```
CTextListBoxModel* lbModel = iListBox->Model();
lbModel->SetItemTextArray(aVideoContainer);
```

Koska videosäiliön omistajuus halutaan säilyttää **CMentorRecDocument**-oliolla, määritellään valintalistalle, että se ei omista sisältötaulukkoa.

```
lbModel->SetOwnershipType(ELbmDoesNotOwnItemArray);
```

Näillä toimilla on varmistettu, että videosäiliön sisältämät videot ovat helposti käsiteltävissä yhtenä yksikkönä, tallennettavissa ja ladattavissa, sekä esitettävissä valintalistan elementteinä.

5.4 Käyttöliittymän ohjelmointi

5.4.1 Ohjelmointitapa

Käyttöliittymän ohjelmoinnissa käytettiin kehitysympäristön UI Designer -käyttöliittymäsuunnittelutyökalua, jolla eri näkymien suunnittelu sujuikin vaivattomasti. Suunnittelutyökalulla suunniteltujen näkymien ohjelmointi on helppoa, koska suunnitelluista näkymistä luodaan suoraan ohjelmakoodia. Suunnittelutyökalu teki kuitenkin ohjelmakoodiin lohkoja, joihin ei voinut tehdä muutoksia. Ajan riittämättömyys UI Designerin ja sen luoman ohjelmakoodin syvälliseen opiskeluun johti siihen, että käyttöliittymä päätettiin ohjelmoida käsin. Näin päästiin itse vaikuttamaan kaikkiin käyttöliittymän ominaisuuksiin ja kontrolloimaan sen toimintaa täysin. Ohjelmoidessaan itse käyttöliittymäkomponentteja oppii myös syvällisemmin ohjelmointikielen ja käyttöjärjestelmän käytäntöjä ja ajattelumalleja.

5.4.2 Käyttöliittymän hallinta

Käyttöliittymän hallintaa varten Carbide.c++ luo projektirunkoon **`CMentorRecAppUi`**-luokan, joka ohjaa käyttöliittymän toimintaa. Käyttöliittymä toteutetaan melko lailla samalla tavalla kaikissa Symbian-sovelluksissa. Koska näkymien vaihtoa ja hallintaa varten otettiin käyttöön S60-alustan näkymänvaihtoarkkitehtuuri, peritään **`CMentorRecAppUi`**-luokka **`CAknViewAppUi`**-luokasta. Näin käyttöliittymää ohjaava luokka osaa käyttää S60-alustan näkymäpalvelinta (view server) näkymien hallintaan. Valmiin käyttöliittymän käytönaikaisia kuvankaappauksia on liitteessä 6.

`CMentorRecAppUi`:n konstruktorissa luodaan näkymät.

```
CMentorRecListView* listView = new(ELeave) CMentorRecListView;
CleanupStack::PushL(listView);
listView->ConstructL(ClientRect());
```

Lisätään näkymät näkymäpalvelimen hallittavaksi näkymäpinoon (view stack).

```
AddViewL(listView);
```

Näkymän id-numero talletetaan **CMentorRecAppUi** jäsenmuuttujaan, jotta voidaan pyytää näkymäpalvelinta vaihtamaan tiettyyn näkymään id-numeron perusteella.

```
iListViewId = listView->Id();
```

Lopuksi määritellään oletusnäkymä, eli näkymä, joka näkyy sovelluksen käynnistyksen jälkeen ensimmäisenä.

```
SetDefaultViewL(*listView);
```

Kun näkymien id-numerot on talletettu jäsenmuuttujiin, voidaan näkymäpalvelimelta pyytää näkymän vaihtoa id-numeroon perustuen. Näin tapahtuu esimerkiksi, kun valikosta valitaan siirtyminen kuvausnäkymään.

```
ActivateLocalViewL(iRecordViewId);
```

Koska eri näkymät tarvitsevat tiedon siitä, mitä videota kulloinkin käsitellään, pitää **CMentorRecAppUi**-luokka myös kirjaa käsittelyssä olevasta videosta tallettamalla sen id-numeron **iSelectedVideoIndex**-jäsenmuuttujaansa. Tämä luokka hoitaa myös näppäinpainallusten sekä komentojen käsittelyn, jos näkymät eivät niitä itse käsittele. Myös sovelluksen tilan tallennus sitä suljettaessa on **CMentorRecAppUi**-luokan vastuulla. Jos saadaan lopetuskomento, kutsutaan **CMentorRecAppUi**-olion **SaveL**-funktiota, joka käynnistää sovelluskehyksen tarjoaman tallennusprosessin.

5.4.3 Näkymäluokat

CMentorRecAppUi-luokan luomat ja näkymäpalvelimelle siirtämät näkymäoliot luodaan **CMentorRecListView**-, **CMentorRecRecordView**- ja **CMentorRecPlayView**-luokista. Näiden luokkien yhteisiä ominaisuuksia voidaan tarkastella käytännössä **CMentorRecListView**-luokan avulla.

CMentorRecListView-luokka peritään **CAknView**:sta, jotta näkymäolio saadaan annettua näkymäpalvelimen hallintaan. Luokka peritään myös rajapintaluokasta **MProgress-**

DialogCallback, jotta sen instanssia voidaan käyttää etenemispalkki-dialogin tapahtumien käsittelyyn. Näkymälle määritellään myös id-numero, jonka tulee olla uniikki käytettävien näkymien joukossa.

```
const TUid KListViewId = {1};
```

Näkymäluokan alaisuuteen luodaan aina myös käyttöliittymän kontrollit sisältävä säiliö (container). Jotta käytössä olevat resurssit saataisiin aktiivisena olevan näkymän käytettäviksi, luodaan kontrollisäiliöolio **iContainer**-jäsenmuuttujaan vasta näkymän aktivoitumisvaiheessa **DoActivateL**-funktiossa. Tässä näkyy myös **CMentorRecDocument**-luokan omistuksessa olevaan videosäiliöön (**videoContainer**) viittaaminen. Videosäiliö annetaan vielä **CMentorRecListContainer**-olion toisen vaiheen konstruktorille (**ConstructL**) käytettäväksi.

```
CVideoContainer* videoContainer = ((CMentorRecDocument*)GetAppUi()
                                                ->Document())->VideoContainer();
iContainer = new(ELeave) CMentorRecListContainer(this);
iContainer->SetMopParent(this);
iContainer->ConstructL(ClientRect(), videoContainer);
```

Voidakseen vastaanottaa näppäinpainalluksia ja siten myös syötettä, lisätään kontrollisäiliö lopuksi **CMentorRecAppUi**:n kontrollipinoon (control stack).

```
AppUi()->AddToStackL(*this, iContainer);
```

DoActivateL-funktiossa kysytään myös **CMentorRecAppUi**-luokalta käsiteltävän videon indeksi, ja siirretään valintalistan osoitin kyseisen videon kohdalle.

```
if (GetAppUi()->SelectedVideoIndex() >= 0)
    {
    iContainer->SetSelectedVideoIndex(GetAppUi()->SelectedVideoIndex());
    }
```

Näkymän poistuessa otetaan ensin talteen valitun videon indeksi ja annetaan se **CMentorRecAppUi**-luokan talletettavaksi. Sitten kontrollisäiliö poistetaan kontrollipinosta ja tuhotaan säiliö.

```
if (iContainer)
    {
    if (iContainer->SelectedVideoIndex() >= 0)
        {
        GetAppUi()->SetSelectedVideoIndex(iContainer->SelectedVideoIndex());
        }
    AppUi()->RemoveFromViewStack(*this, iContainer);
    }
delete iContainer;
iContainer = NULL;
```

5.4.4 Kontrollisäiliöluokat

Kontrollisäiliöluokkia ovat näkymiä vastaavasti: **CMentorRecListContainer**, **CMentorRecRecordContainer** ja **CMentorRecPlayContainer**. Näihin säiliöihin sisällytetään kaikki näkymän sisältämät kontrollit. Säiliöluokat peritään kaikkien kontrollien kantaluokasta **CCoeControl**ista, joten kontrollisäiliö on itsessäänkin kontrolli. Kontrollisäiliöt toimivat periaatteessa hyvin samankaltaisesti, eroa syntyy vain niiden sisältämistä kontrolleista. Periaatteita valottamaan luodaan katsaus **CMentorRecListContainer**-luokan toimintaan.

Kontrollisäiliötä luotaessa tulee jäsenmuuttujaan tallettaa osoitin säiliön omistavaan näkymään.

```
CMentorRecListContainer::CMentorRecListContainer(CAknView* aOwningView)
: iOwningView(aOwningView)
    {
    }
```

Jotta näppäimenpainallukset ja kohdistimen siirrot toimisivat kontrollisäiliön sisällä, pitää toteuttaa myös funktiot **CountComponentControls** ja **ComponentControl**, joita käytetään kontrollipinon kautta. Ensimmäisen funktion tehtävänä on vain laskea säiliön sisältämät kontrollit. Toinen palauttaa osoittimen sisällytettyyn kontrolliin annetulla indeksillä. Tämän lisäksi kontrollisäiliöluokista instantioitaessa luodaan säiliön sisältämät kontrollit. Kontrollit täytyy myös poistaa säiliötä tuhottaessa.

5.5 Tietoliikenneyhteys

5.5.1 HTTP-kehyksen toiminta

Symbian-käyttöjärjestelmä tarjoaa HTTP-yhteyksiä varten protokollapinon, jota kutsutaan HTTP-kehykseksi. Kehys tarjoaa kaikki protokollan määrittelemät metodit, mukaan lukien Get ja Post. Protokollan toiminnasta ei sovellusohjelmoijan tarvitse tietää mitään, sillä HTTP-kehys tarjoaa käytettävät rajapinnat sovellusohjelmointiin. Kaikki protokollan määrittelemät toiminnot suoritetaan sovellusohjelmoijan näkymättömissä. HTTP-kehys voi muodostaa istuntoja, vaikka HTTP onkin tilaton protokolla. Istunnon aikana tapahtuva tiedonvaihto jaetaan siirtotapahtumiin (transaction), joita voi olla käynnissä useita samanaikaisesti. Tiedonsiirto suoritetaan asynkronisesti, kutsumalla tiedonvaihdon aloittavaa funktiota. Tämän jälkeen toimintaa ohjaa HTTP-tapahtumien tarkkailija. Tarkkailija saa tiedon aina, kun jotakin merkittävää tapahtuu siirtoyhteyteen liittyen. Merkittäviä tapahtumia ovat esimerkiksi datapaketin vastaanottaminen ja siirron onnistuminen tai epäonnistuminen. Tarkkailijan vastuulla on tällöin suorittaa tarvittavat toimet tapahtuman laadusta riippuen. HTTP-kehyksen käyttöä varten on määriteltävä sovellus käyttämään NetworkServices-kyvykkyyttä. [23, s. 343–361].

5.5.2 Yhteyden toteutus

HTTP-yhteyksiä hoitamaan kirjoitettiin **`CHttpClient`**-luokka sekä sille tarkkailijarajapinta **`MHttpClientObserver`**. Luokan **`CHttpClient`** toteutus on liitteessä 3.

Kun **`CHttpClient`**-olio luodaan videolistanäkymässä, tehdään videolistanäkymän **`CMentorRecListContainer`**-kontrollisäiliöstä HTTP-asiakkaan tarkkailija.

```
iClient = CHttpClient::NewL(*iContainer);
```

Annetaan **`CHttpClient`**-olion tietoon, mikä näkymä sen omistaa.

```
iClient->SetCallBack(this);
```

Konstruoitaessa **CHttpClient**-oliota avataan istunto käyttäen oletusyhteyskäytäntöä HTTP/TCP. Ennen istunnon avaamista on kyseisen istunnon oltava suljettu ja aktiivisen vuorottajan (active scheduler) oltava asennettuna. [24.]

```
iSession.OpenL();
```

Määritetään **CHttpClient**-luokka vastaamaan todennuspyyntöihin.

```
InstallAuthenticationL(iSession);
```

Asetetaan käytettävä HTTP-metodi ja avataan virtuaalinen HTTP-yhteys. Yhteyttä ei aidosti ole olemassa, koska HTTP on yhteydetön protokolla. Get- ja Post-metodit toimivat samalla periaatteella. Esimerkiksi lähetettäessä tietoa palvelimelle tarvitaan Post-metodia, johon tässä keskitymme.

```
RStringF method = iSession.StringPool().StringF(HTTP::EPOST,
                                                RHTTPSession::GetTable());
iTransaction = iSession.OpenTransactionL(uri, *this, method);
```

Asetetaan Sovelluksen tiedot ja vastaanotettavat sisältötyypit lähetettävään tietoon. Lisäksi lähetettäessä dataa määritellään myös sisältötyyppi (content type).

```
RHTTPHeaders headers = iTransaction.Request().GetHeaderCollection();
SetHeaderL(headers, HTTP::EUserAgent, KUserAgent);
SetHeaderL(headers, HTTP::EAccept, KAccept);
SetHeaderL(headers, HTTP::EContentType, aContentType);
```

Asetetaan tietojen tarjoajaksi tämä **CHttpClient**-luokan ilmentymä. Näin vastaanottajan pyytäessä lisää datapaketteja ohjataan pyynnöt HTTP-asiakasluokan oliolle, joka hoitaa lisäpakettien lähettämisen.

```
MHTTPDataSupplier* dataSupplier = this;
iTransaction.Request().SetBody(*dataSupplier);
```

Käynnistetään tiedonvaihto ja asetetaan jäsenmuuttujaan tieto tiedonsiirron käynnistymisestä.

```
iTransaction.SubmitL();
iRunning = ETrue;
```

Lisäksi asetetaan vielä omistavaan näkymään näkyville odotusilmoitus tiedonsiirron ajaksi.

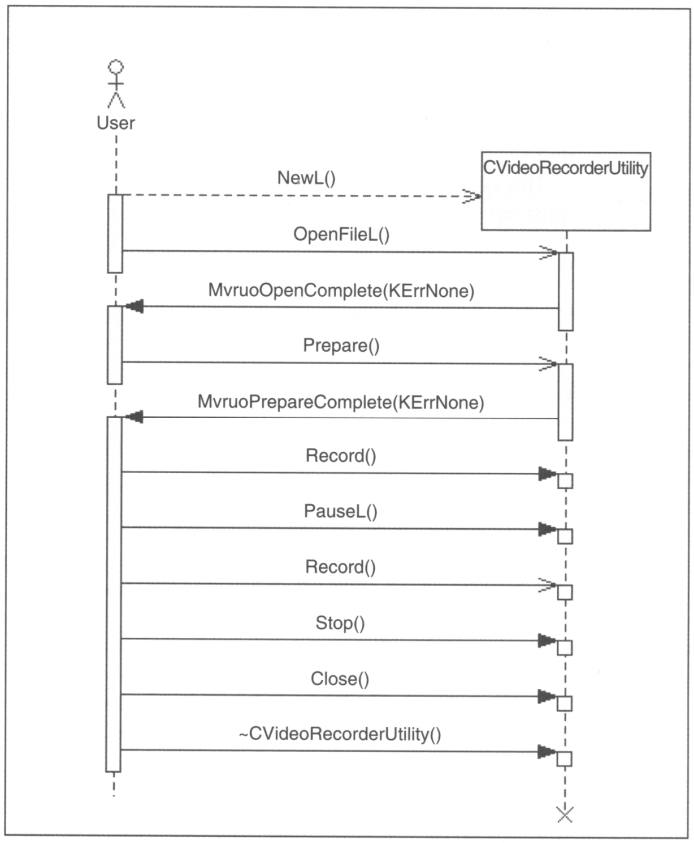
CHttpClient-luokan **MHFRunL**-funktiota kutsutaan aina, kun tapahtuu jotain merkittävää tiedonsiirron kannalta. Tapahtumia ovat esimerkiksi lähetyksen valmistuminen, virhe lähetyksessä sekä datapaketin vastaanottaminen. Riippuen tapahtuman luonteesta suoritetaan tilanteeseen sopivin toiminta ja ilmoitetaan tarkkailijalle. Jos saatiin palvelimelta osa sen lähettämää dataa, annetaan sekin tarkkailijalle käsiteltäväksi.

Virheen sattuessa tiedonsiirtoprosessissa käsitellään se virheen laadusta riippuen **MHFRunError**-funktiossa asiaankuuluvalla tavalla.

5.6 Videon kuvaaminen ja tallentaminen

5.6.1 Kameran käyttö Symbian-käyttöjärjestelmässä

Videon kuvaukseen pääsee Symbian-käyttöjärjestelmässä käsiksi multimediakehyksen tarjoamien rajapintojen avulla. Videon kuvausta ja tallennusta varten tarvitaan videonkäsittelyyn tarkoitettua sovellusohjelmointirajapintaa. Videota voi tallentaa kuvattaessa tiedostoon, tietovirtana määriteltyyn URL-osoitteeseen (uniform resource locator) tai muistiin. Ohjelmoija määrittelee käytettävän kameran, tiedostomuodon ja käytettävän pakkausstandardin. Videon kuvaaminen tapahtuu asynkronisesti, muuta puhelimen toimintaa häiritsemättä. Kuvassa 5 nähdään, kuinka tyypillinen kuvausprosessi etenee tarkkailijan ohjauksessa. Kameran käyttöön vaaditaan sovellukselta UserEnvironment-kyvykkyys.



Kuva 5. Tapahtumien kulku videota kuvattaessa [9, s. 89.]

5.6.2 Kameran valmistelu

Kameran käyttöä varten kirjoitettiin luokka **`CCameraEngine`**. Tämän luokan kautta on tarkoitus ohjata järjestelmän **`CCamera`**-luokan operaatioita. **`CCameraEngine`**-luokka toteuttaa **`MCameraObserver`**-rajapinnan **`CCamera`**:n operaatioiden seuraamista varten. Luotiin myös **`MCameraEngineObserver`**-luokka tarjoamaan **`CCameraEngine`**-oliolle tarkkailurajapinta. Tämän rajapinnan tässä sovelluksessa toteuttaa **`CMentorRecRecordView`**-luokka, joka oli selkeä valinta tarkkailijaksi, sillä vain kuvausnäkymässä käytetään kameraa. **`CCameraEngine`**-luokan ohjelmakoodi on liitteessä 4.

Luotaessa **`CCameraEngine`**-oliota asetetaan referenssijäsenmuuttuja **`iObserver`** osoittamaan tarkkailijaolioon tapahtumien seuraamista varten. Samoin talletetaan referenssi **`CCoeControl`**-tyyppiseen kontrolliin, johon kameran etsimen kuva tulostetaan.

```
CCameraEngine::CCameraEngine(MCameraEngineObserver& aObserver,
                             const CCoeControl& aControl)
: iObserver(aObserver),
  iControl(aControl)
    {
    }
```

Käyttöön halutaan kameralaite, joka osoittaa käyttäjästä poispäin ja on useampia kameroita sisältävissä matkapuhelimissa yleensä korkeampierottelukykyinen kamera. [25, s. 256]. Tätä varten on käytävä läpi kameralaitteita, kunnes löytyy poispäin osoittava kamera.

```
for (TInt i = 0 ; i < CCamera::CamerasAvailable() ; i++)
```

Luodaan **CCamera**-olio, jonka tarkkailijaksi asetetaan **CCameraEngine**-olio. Tarkastetaan, osoittaako kamera käyttäjästä poispäin.

```
cam = CCamera::NewL(*this, i);
cam->CameraInfo(camInfo);
if (camInfo.iOrientation == TCameraInfo::EOrientationOutwards)
```

Valitaan ensimmäinen ulospäin osoittava kamera ja otetaan se jäsenmuuttujaan talteen. Talteen otetaan myös kameran **TcameraInfo**-olio.

```
iCamera = cam;
iCameraInfo = camInfo;
```

Samassa silmukassa valitaan vielä oikea etsimen toimintatila testaamalla **iCameraInfo**-olion **iOptionsSupported**-muuttujaa biteittäin. Välittömän etsintilan testaaminen on sinänsä S60-laitteessa turhaa, koska S60-laitteet eivät tällä hetkellä tue välitöntä etsintilaa (direct viewfinder mode). [25, s. 257.] Tulevaisuutta varten kirjoitettiin kuitenkin testi myös välitöntä etsintilaa varten.

```
if ( ( iCameraInfo.iOptionsSupported &
       TCameraInfo::EViewFinderDirectSupported ) > 0)
    {
    iViewFinderMode = EViewFinderDirect;
    }
else if ( ( iCameraInfo.iOptionsSupported &
            TCameraInfo::EViewFinderBitmapsSupported ) > 0)
    {
    iViewFinderMode = EViewFinderBitmap;
    }
else
    {
    iViewFinderMode = EViewFinderNone;
    }
```

Kuvausnäkymän aktivoituessa käynnistyy komentoketju, jonka seurauksena kameran etsimen kuva näytetään ruudulla. Tässä yhteydessä määritellään näytettävän etsimen koko. Kooksi pitäisi asettaa yleisesti etsimen kuvan näyttävän kontrollin koko. Koon asettaminen ei kuitenkaan onnistunut näin, joten käytettiin prototyypissä laitekohtaista kovakoodattua kokoa. Lopuksi käynnistetään kameran varaaminen **`Reserve`**-funktiolla.

```
if (iViewFinderMode == EViewFinderNone)
    {
    iObserver.OnError(KErrNotSupported);
    }
else
    {
    iViewFinderSize = TSize(200,240); // iControl.Size();
    iCamera->Reserve();
    }
```

Kameran varaaminen toimii asynkronisesti, ja **`CCamera`**-olio kutsuu valmistuttuaan tarkkailijan **`ReserveComplete`**-funktiota [25, s. 257]. Tämän funktion päätehtävä on pyytää **`CCamera`**-luokan oliota kytkemään kameran virta päälle, mikäli kameran varaaminen sujui ongelmitta.

```
iCamera->PowerOn();
```

Myös **`PowerOn`** on asynkroninen funktiokutsu. Sen valmistuminen johtaa tarkkailijan **`PowerOnComplete`**-funktion kutsuun. Funktiossa pyydetään **`CCamera`**-oliota käynnistämään kameran etsin.

```
iCamera->StartViewFinderBitmapsL(iViewFinderSize);
```

Jos virran kytkemisessä tapahtui virhe, vapautetaan kameran varaus ja ilmoitetaan tarkkailijalle.

```
iCamera->Release();
iObserver.OnError(aError);
```

Aina kun uusi etsimen kuva on valmiina piirrettäväksi, kutsutaan **CCamera**-luokan tarkkailijan funktiota **ViewFinderFrameReady** [25, s. 258]. Funktion toteutuksessa tärkeimpänä toimena kutsutaan kamera-oliota tarkkailevan kuvausnäkymäolion funktiota **OnViewFinderFrameReady** ja annetaan parametrina **CCamera**-oliolta saatu etsimen kuva. Jää tarkkailijan vastuulle näyttää tämä kuva näytöllä. Tärkeätä on myös nollata toimettomuusajan laskuri (inactivity time), jotta taustavalo pysyisi päällä ja kuvan voisi näytöltä nähdä myös hämärissä olosuhteissa.

```
User::ResetInactivityTime();
iObserver.OnViewFinderFrameReady(aFrame);
```

5.6.3 Videon kuvaaminen

Videon kuvaamiseen tarvittiin oma luokka, **CMentorVideoRecorder**. Se käyttää **CVideoRecorderUtility**-luokkaa videon kuvaamiseen. Videon kuvausluokka toteuttaa **MVideoRecorderUtilityObserver**-rajapinnan, jotta se voi toimia videon kuvauksen hoitavan olion tarkkailijana. Luokan ohjelmakoodilistaus on liitteessä 5.

CMentorVideoRecorder-olion konstruktorissa luodaan ensin **CVideoRecorderUtility**-olio.

```
iVideoRecorderUtility = CVideoRecorderUtility::NewL(*this);
```

Lisäksi määritellään video-ohjaimen (controller) ja tallennusmuotojen asetuksia sekä valitaan käytettävä video-ohjain kuvausta varten. Sovellus odottaa komentoa alkaa ku-

vata videota näyttäen samalla etsimen kuvaa näytöllä. Kuvaus alkaa keskimmäisestä valintanäppäimestä tai valikon kohdasta "Kuvaa". Kuvauskäskyn saatuaan sovellus pyytää avattavaksi tiedoston, johon uusi video kuvattaessa tallennetaan.

```
iVideoRecorderUtility->OpenFileL(aFileName, aCameraHandle,
                                 iControllerUid, iFormatUid);
```

Tiedoston avaaminen on asynkroninen operaatio, jonka päättyessä kutsutaan **CVideoRecorderUtility**-oliota tarkkailevan **CMentorVideoRecorder**-olion **MvruoOpenComplete**-funktiota. Asetetaan videoleikkeen maksimikoko ja äänen tallennuksen tila. Jos nämä toimet onnistuvat, käynnistetään videotallentimen valmistelu.

```
TRAPD(error,
    iVideoRecorderUtility->SetMaxClipSizeL(KMaxVideoSize);
    iVideoRecorderUtility->SetAudioEnabledL(ETrue);
    );
if (aError == KErrNone)
    {
    iVideoRecorderUtility->Prepare();
    }
```

Prepare-funktion suorituksen päättyminen aiheuttaa kutsun **MvruoPrepareComplete**-funktioon. Käynnistetään tallennus.

```
iVideoRecorderUtility->Record();
```

Tallennus pysäytetään samoin kuin käynnistettiinkin, eli kun käyttäjä valitsee valikosta "Pysäytä" tai painaa keskimmäistä valintapainiketta. Näytössä näkyy ilmoitus tallennuksen päättymisestä.

```
iVideoRecorderUtility->Stop();
iVideoRecorderUtility->Close();
```

5.6.4 Lopputoimet

Video on tallennettu, mutta kamerassa on vielä virta päällä ja se on varattuna käyttöön. Poistuttaessa kuvausnäkymästä Poistetaan **`CMentorVideoRecorder`**-olio, joka sulkee ja poistaa myös videon tallennusolion.

```
iVideoRecorderUtility->Close();
```

Sitten kutsutaan **`CCameraEngine`**-luokan **`StopViewFinderL`**-funktiota, jossa poistetaan kameran etsin näytöltä. Kamera vapautetaan varauksesta. Vapautus katkaisee myös kameran virran.

```
iCamera->StopViewFinder();
iCamera->Release();
```

Kameraa käytettäessä on tärkeää muistaa, että käynnissä oleva kamera kuluttaa paljon virtaa. On siis tärkeää vapauttaa kamera ja katkaista siitä virta, jos sitä ei käytetä. [9, s. 39–40].

6 Katsaus sovelluskehitysprosessiin

6.1 Käytännön ongelmia

Työn tarkoituksena oli luoda helppokäyttöinen, missä vain toimiva opetusvideon kuvausratkaisu paikkatiedon tallennuksen kera. Yltiöpäisistä suunnitelmista päädyttiin karsimaan kehitysprosessin edetessä suuri osa ja tekemään lopputuotteesta vain hyvin pelkistetty prototyyppi. Prototyypin valmiiksi saaminenkin estyi lopulta taloustilanteen aiheuttaman pakon sanelemana. Sovellukseen suunniteltiin aluksi GPS-paikannus, joka päätettiin pian jättää ensimmäisestä prototyyppiversiosta pois. Optimistinen aikataulu ei pitänyt, ja suunnitelluista ominaisuuksista ehdittiin saada täysin valmiiksi ainoastaan videoiden hallinta ja käyttöliittymä. Tietoliikenneyhteydet toteutettiin suunnitellusti, mutta kunnollista testausta ei niille vielä ole tehty. Kameran käyttö suunniteltiin perusteellisesti ja se myös ohjelmoitiin. Kuva kamerasta saatiin näytölle, muttei vielä tallen-

nettua. Tallennusluokka kirjoitettiin kokonaisuudessaan, mutta virheenetsintä jäi kesken projektin päättyessä. Prototyyppiä varten olisi siis vielä toteutettava loppuun videon kuvaus sekä testattava koko sovellus hyvin.

6.2 Tuotteen käyttökelpoisuus

Tuote sopisi valmiiksi saatettuna hyvinkin käyttötarkoitukseensa toiminnallisuutensa ja helppokäyttöisyyden ideologiansa puolesta. Esteeksi voisivat nousta puhelinmallien erilaisuus käytännön ominaisuuksissa. Joissakin puhelimissa on tarkkaan erotteluun pystyvä kamera, kun toisissa on heikompi kuvanlaatu. Kuvanlaadun varmistaminen ja alimman hyväksyttävän erottelukyvyn löytäminen vaatisi mittavaa testausta eri puhelinmalleilla. Myös näytön koko saattaisi rajoittaa käyttöön hyvin soveltuvien matkapuhelinten valikoimaa. Sovellus olisi käytettävissä pienelläkin näytöllä, mutta käytettävyys huononisi. Uusissa S60-älypuhelimissa näytön koko tulee tuskin olemaan ongelma isohkojen näyttöjen ja pienemmälläkin näytöllä suuremman näytön erottelukyvyn ansiosta.

MentorAid-tuotteen myyntiin videon kuvaussovelluksella voisi olla positiivinen vaikutus. Samankaltaista videon tuottamisratkaisua on tuskin kilpailevilla toimijoilla tarjota. Matkapuhelimella tapahtuvan videon tuottamisen konsepti pitäisi vain ensin saada luotettavaan ja asialliseen pakettiin. Siten matkapuhelimiin työkaluna skeptisestikin suhtautuvat saataisiin vakuuttuneiksi helppokäyttöisen ratkaisun kannattavuudesta.

6.3 Jatkokehitys

6.3.1 Parannuksia ja viimeistelyä

Jotta videon tallennussovelluksesta voisi tulla oikea ohjelmistotuote, olisi koko sovelluksen toiminta vielä tarkastettava. Toiminnan kulusta olisi löydettävissä vielä paljon korjattavaa. Lopputaipaleella kehitystyössä tuli muun muassa esille, että kameran etsimen ollessa puhelimen näytöllä kuluu akusta paljon virtaa. Se lienee hyväksyttävää, kun

kuvataan, mutta kuvaustoimintoon tulisi lisätä etsimen näytöltä poistava ja kameran virran pois kytkevä viivekytkin. Näin akun turha tyhjeneminen voitaisiin minimoida. Myös yllättävät tilanteet, kuten saapuva puhelu tai akun varauksen loppuminen, tulisi käsitellä siten, ettei tietoja menetetä. Myös epälooginen toiminta tulisi voida välttää esimerkiksi sovelluksen joutuessa puhelua vastaanotettaessa taka-alalle ja sen palatessa taas näkyviin. Videon kuvaaminen tulisi tietysti pysäyttää puhelun ajaksi, ja sen säilyä pysäytettynä puhelun päätyttyä. Sovellus voi toimia odottamattomasti myös, kun herätyskello soi tai kameran luukku suljetaan. Videon kuvaamisen toimintaketju pitäisi myös viimeistellä, sillä esimerkiksi videoinnin pysäytystä kesken kuvauksen ei vielä toteutettu. HTTP-yhteys pitäisi toteuttaa HTTPS-protokollaa käyttäen, sillä tietoliikenne ohjelmoitiin nyt käyttämään salaamatonta HTTP-yhteyttä.

6.3.2 Lisäominaisuudet

Sovellukseen pitäisi ehdottomasti toteuttaa jo alun perin suunniteltu asetusnäkymä. Pitäisihän sovelluksen käyttäjän voida muuttaa vaikkapa videon lähetysasetuksia. GPS-paikannus on myös yksi kuvaussovelluksen alkuperäisistä vaatimuksista. Paikannus oli itse asiassa yksi tärkeimpiä, koska se lisää helppokäyttöisyyttä asettamalla videolle automaattisesti kuvauspaikkamääreen käsin syöttämisen sijaan.

Käytettävyyden ja toimivuuden kannalta voisi olla hyvä tutkia tulevaisuudessa vaihtoehtoja saada videon lähetys käynnistettyä vasta vaikkapa yrityksen omassa langattomassa lähiverkossa. Näin saataisiin tiedonsiirtonopeus suureksi, eikä pakotettaisi käyttäjää odottelemaan hyvinkin pitkiä aikoja videon siirtoa. Langatonta verkkoa voisi varmasti myös käyttää paikannukseen sisätiloissa. WLAN-paikannuksen toteuttaminen näin pienlevikkiseen sovellukseen lienee kuitenkin, ainakin näillä näkymin, liian suuritöistä.

Videon kuvaussovelluksen suunnitteluvaiheessa nousi myös esiin ajatus siitä, että voisihan opetusvideoita katsellakin mobiililaitteella. Yksi hyvä jatkokehityksen suunta voisi olla luoda myös katselusovellus, joka voisi vaikkapa hakea opastevideot, jotka liittyvät laitteen tämänhetkiseen sijaintiin. Sijainti saataisiin tietysti GPS-laitteelta, kuten kuvaussovelluksessakin. Näin saataisiin myös opiskelu mobilisoitua.

6.4 Testaus ja käyttöönotto

Ennen tuotteen käyttöönottoa olisi lisäominaisuuksien ohjelmoimisen lisäksi suoritettava mittavaa ja järjestelmällistä testausta. Testaussuunnitelma tulisi laatia ja siihen sisällyttää sekä yksikkö- että integraatiotestausta. Testauksen edettyä riittävän pitkälle, suoritettaisiin tuotteen korjaustoimet. Kun todettaisiin sovellus riittävän toimivaksi, voitaisiin tuote saattaa asiakkaiden kokeiltavaksi. Näiden kokeilujen pohjalta olisi kerättävä monipuolista palautetta peruskäyttäjiltä, jotta tuotetta osattaisiin kehittää eteenpäin käyttäjän näkökulmasta. Sovellus tulisi myös hyväksyttää ennen varsinaista käyttöönottoa. Ennen hyväksytystesteihin toimittamista koko sovellus tulisi testata itse Symbian Signed -testauksen kriteerein. Käyttöönottoon tulisi kuulumaan koulutusta sovellusta käyttäville asiakasyrityksille sekä Mamentor Oy:n henkilökunnalle tukitoimintojen varmistamiseksi.

7 Yhteenveto

Työn tarkoituksena oli suunnitella ja toteuttaa opetusvideoiden kuvaamiseen ja siirtoon erikoistunut matkapuhelinsovellus. Sovelluksen suunnittelussa oli käytettävyys keskeisessä osassa, sillä tuotteen tuli olla kaikentasoisten käyttäjien käytettävissä. Valmistettavan prototyyppisovelluksen piti toimia alustana, jolla kokeillaan saako opetusvideoita järkevästi kuvattua matkapuhelimella. Kuvaamisen lisäksi myös videon siirtoa oli määrä tarkastella ja arvioida, ovatko videonsiirron kaistanleveysvaatimukset mobiililaitteille liian suuret. Lopullista prototyyppiä ei kuitenkaan saatu valmiiksi, koska projekti keskeytettiin heikentyneen taloudellisen tilanteen takia.

Suunnittelu- ja ohjelmointityön tuloksia voitaneen käyttää hyväksi, jos projektia päätetään taloudellisen tilanteen salliessa jatkaa. Videon kuvaus onnistuu riittävällä tarkkuudella ainakin osassa markkinoilla olevista matkapuhelimista. Tiedonsiirtoa ei päästy kokeilemaan varsinaisilla videotiedostoilla, joten videoiden siirtoajoista ja siirron toimivuudesta ei vielä ole faktatietoa. Sovelluksen suunnittelun ja ohjelmoinnin aikana tulleet ajatukset toivat eniten innovatiivisia, uusia ideoita. Sovellukseen olisi nyt jo monta kehitysideaa sekä sovelluksen ominaisuuksia että ohjelmistokehitysratkaisuja koskien.

Videon tallennussovelluksesta voidaan saada MentorAid-osaamisenhallintajärjestelmään hyödyllinen jatke. Opetusvideoiden kuvaaminen voitaisiin suorittaa käytännön työtilanteissa ja minimaalisella laitteistolla. Tästä saataisiin etua kilpailijoihin nähden, sillä vastaavaa ominaisuutta ei samankaltaisissa tuotteissa ole yleensä tarjolla.

Lähteet

1 The Global Positioning System. (WWW-dokumentti.) The National Executive Committee for Space-Based Positioning, Navigation, and Timing. <http://www.gps.gov/systems/gps/index.html>. 30.6.2008. Luettu 12.4.2009.

2 Wiegand, Thomas & Sullivan, Gary J. & Bjøntegaard, Gisle & Luthra, Ajay: Overview of the H.264/AVC Video Coding Standard. (WWW-dokumentti.) IEEE. <http://ip.hhi.de/imagecom_G1/assets/pdfs/csvt_overview_0305.pdf> 7.7.2003. Luettu 12.4.2009.

3 Bleidt, Robert: Understanding MPEG-4: Technologies, Advantages, and Markets. (WWW-dokumentti.) The MPEG Industry Forum. <http://www.m4if.org/public/documents/vault/MPEG4WhitePaperV2a.zip>. 2005. Luettu 12.4.2009.

4 Java Server Pages Technology. (WWW-dokumentti.) Sun Microsystems, Inc. <http://java.sun.com/products/jsp/> 11.5.2005. Luettu 12.4.2009.

5 Python for S60. (WWW-dokumentti.) Nokia Research Center. <http://wiki.opensource.nokia.com/projects/PyS60>. 1.4.2009. Luettu 12.4.2009.

6 RFID-tietoutta. (WWW-dokumentti.) RFIDLab Finland. <http://www.rfidlab.fi/?1;2;800;800.html>. 21.8.2007. Luettu 7.4.2009.

7 Symbian OS. (WWW-dokumentti.) Wikipedia. <http://fi.wikipedia.org/wiki/Symbian_OS>. 20.3.2009. Luettu 12.4.2009.

8 TCP. (WWW-dokumentti.) Wikipedia. <http://fi.wikipedia.org/wiki/TCP>. 9.3.2009. Luettu 12.4.2009.

9 Rome, Adi & Wilcox, Mark: Multimedia on Symbian OS - Inside the Convergence Device. Chichester: John Wiley & Sons Ltd. 2008.

10 Extensible Markup Language (XML) (WWW-dokumentti.) The World Wide Web Consortium (W3C). <http://www.w3.org/XML/>. 5.4.2009. Luettu 8.4.2009.

11 Mikkonen, Tommi: Programming Mobile Devices : An Introduction for Practitioners. Chichester: John Wiley & Sons, Ltd. 2007.

12 Mason, Sam & Korolev, Elise : Native and Java ME Development on Symbian OS. (WWW-dokumentti.) Symbian Developer Network. <http://developer.symbian.com/main/downloads/papers/Native_And_Java_ME_Dev_On_SymbianOS_%20v1.1.pdf>. 2008. Luettu 26.10.2008.

13 Java™ Verified Program and Unified Testing Initiative. (WWW-dokumentti.) Sun Microsystems, Inc. <http://java.sun.com/javame/reference/docs/java_verified_program.pdf>. 2007. Luettu 26.10.2008.

14 Capability granting process. (WWW-dokumentti.) Nokia. <http://www.forum.nokia.com/main/technical_services/testing/cap_granting.html>. 2008. Luettu 17.11.2008.

15 Piirainen, Tero: Flowplayer technical facts. (WWW-dokumentti.) Flowplayer Ltd. <http://flowplayer.org/documentation/technical-facts.html>. 2008. Luettu 10.12.2008.

16 Multimedia Framework Architecture in S60 Devices. (WWW-dokumentti.) Nokia. <http://sw.nokia.com/id/47cc7ad7-eb00-4bc3-836a-bcf0ebdea51a/Multimedia_Framework_Architecture_in_S60_Devices_v1_1_en.pdf>. 8.5.2007. Luettu 10.12.2008.

17 Global smart phone shipments rise 28%. (WWW-dokumentti.) canalys.com. <http://www.canalys.com/pr/2008/r2008112.htm>. 6.11.2008. Luettu 11.12.2008.

18 Pesonen, Ville & Heikkilä, Jokke & Niittynen, Kimmo. Symbian ohjelmointi 1. Harjoitustyön raportti. Metropolia Ammattikorkeakoulu, 2008. Materiaali tekijän hallussa.

19 S60 platform evolution. (WWW-dokumentti.) Nokia. <http://library.forum.nokia.com/index.jsp?topic=/S60_3rd_Edition_Cpp_Developers_Library/GUID-E77111FD-191E-4C6A-BD41-E9E44CDB353A.html>. 2008. Luettu 11.12.2008.

20 Framework requirements for GUI applications. (WWW-dokumentti.) Nokia. <http://library.forum.nokia.com/index.jsp?topic=/S60_3rd_Edition_Cpp_Developers_Library/GUID-BDDDF68F-F7C3-43AF-8B6C-C77C701FD2A9.html>. 2008. Luettu 11.12.2008.

21 S60 view architecture. (WWW-dokumentti.) Nokia. <http://library.forum.nokia.com/index.jsp?topic=/S60_3rd_Edition_Cpp_Developers_Library/GUID-68B999C2-0993-4804-9624-42C3D88BE5C7.html>. 2009. Luettu 13.4.2009.

22 Laine, Hannu: Symbian ohjelmointi 2. Luentomoniste. Metropolia Ammattikorkeakoulu, 4.12.2008.

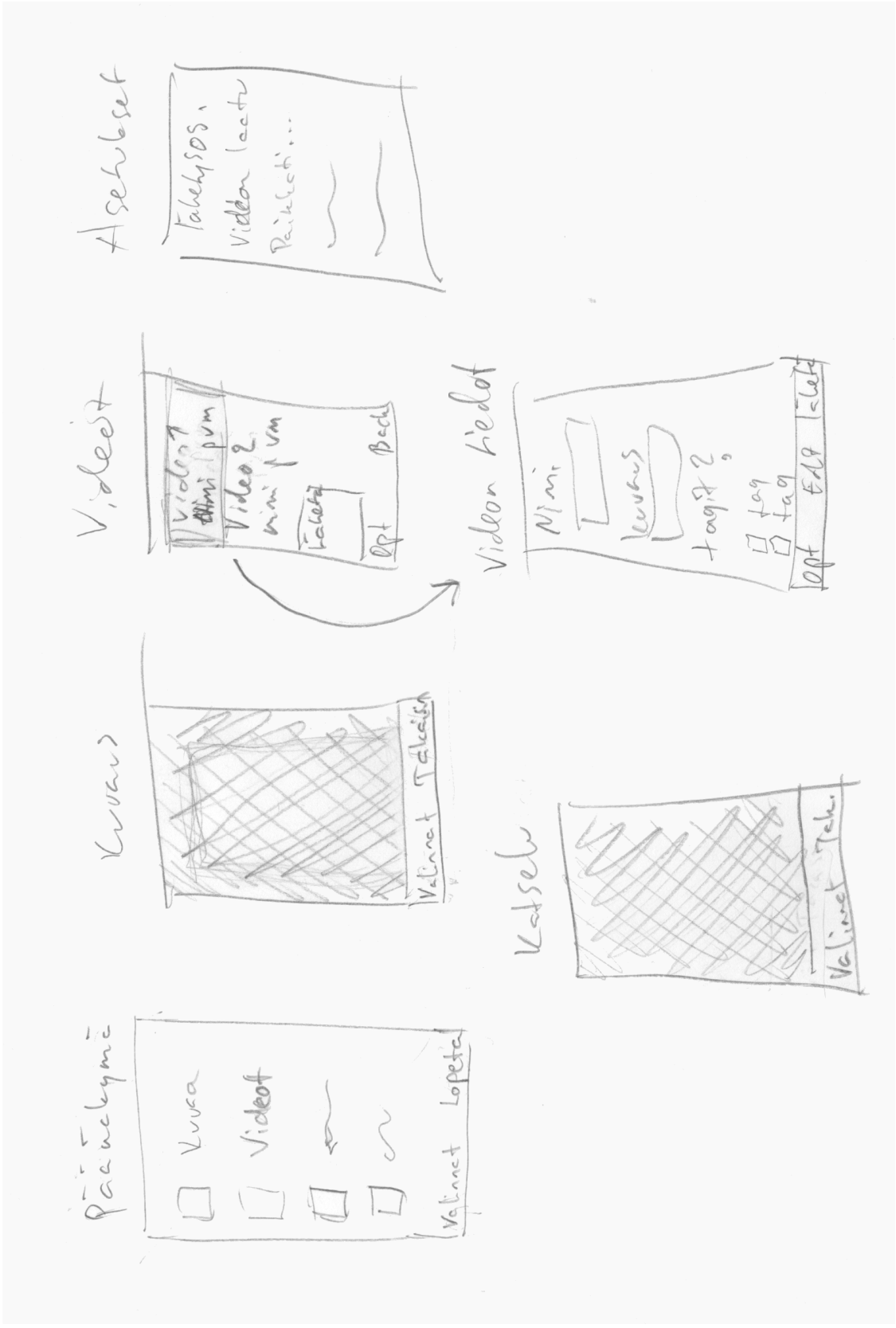
23 Iain Campbell: Symbian OS Communications Programming, Second Edition. Chichester: John Wiley & Sons, Ltd. 2007.

24 Class RHTTPSession. (WWW-dokumentti.) Symbian Software Ltd. <http://www.symbian.com/developer/techlib/v9.2docs/doc_source/reference/reference-cpp/HTTP/RHTTPSessionClass.html>. 2007. Luettu 5.4.2009.

25 Aubert, Michael: Quick Recipes on Symbian OS - Mastering C++ Smartphone Development. Chichester: John Wiley & Sons Ltd. 2008.

```
import appuifw
import e32
import urllib
import re

# set the screen size to large
appuifw.app.screen='large'

tempfile = None

# function definitions

def cls():
  i = 0
  while (i < 20):
    print u"\n"
    i = i+1

def fetchfile():
    global tempfile
    # define a url where what you want to download is located on the net
    url = "http://weather.yahooapis.com/forecastrss?p=FIXX0001&u=c"
    # define the file name and the location of the downloaded file for local
storage e.g. on the c drive
    tempfile = "c:\\python\\weather.xml"
    try:
        # fetch the data
        urllib.urlretrieve(url, tempfile)
    except:
        print "Could not fetch file."

cls()
fetchfile()

#print what we want

fileHandle = open (tempfile)
data = fileHandle.read()
fileHandle.close()

conStart = data.find("<yweather:condition")
conEnd = data.find("/>", conStart)

textStart = data.find("text=\"", conStart) + len("text=\"")
textEnd = data.find("\"", textStart)
tempStart = data.find("temp=\"", conStart) + len("temp=\"")
tempEnd = data.find("\"", tempStart)
print u"\nEspoo now: " + data[textStart:textEnd] + u", " +
data[tempStart:tempEnd] + u" C"

f1Start = data.find("<yweather:forecast")
f1End = data.find("/>", f1Start)
f1TextStart = data.find("text=\"", f1Start) + len("text=\"")
f1TextEnd = data.find("\"", f1TextStart)
f1Text = data[f1TextStart:f1TextEnd]
```

```
f1LoTempStart = data.find("low=\"", f1Start) + len("low=\"")
f1LoTempEnd = data.find("\"", f1LoTempStart)
f1LoTemp = data[f1LoTempStart:f1LoTempEnd]

f1HiTempStart = data.find("high=\"", f1Start) + len("high=\"")
f1HiTempEnd = data.find("\"", f1HiTempStart)
f1HiTemp = data[f1HiTempStart:f1HiTempEnd]

f1DayStart = data.find("day=\"", f1Start) + len("day=\"")
f1DayEnd = data.find("\"", f1DayStart)
f1Day = data[f1DayStart:f1DayEnd]


print u"\nEspoo on " + f1Day + u": " + f1Text + u", " + f1LoTemp + u" C - " +
f1HiTemp + u" C"

running=1


# 5. create and set an exit key handler
def quit():
    global running
    running=0

appuifw.app.exit_key_handler=quit

# set the application title
appuifw.app.title = u"weather"

# create a main loop
while running:
    e32.ao_yield()
```


Päänäkymä
Videot
Valinnat
Videot
Videon tiedot
Kuvaus
Katselu

```
/*
 ============================================================================
 Name            : HttpClient.h
 Author      : kimmonii
 Description : CHttpClient declaration
 ============================================================================
 */

#ifndef HTTPCLIENT_H
#define HTTPCLIENT_H

// INCLUDES
#include <e32std.h>
#include <e32base.h>
#include <http.h>
#include <http\mhttpauthenticationcallback.h>
#include <es_sock.h>    // RSocketServ, RConnection

_LIT8( KUserAgent, "HttpClient 1.0" );
_LIT8( KAccept, "*/*" );

class CMentorRecListView;

class MHttpClientObserver
    {
public:
    virtual void ClientEvent(const TDesC& aEvent) = 0;
    virtual void ClientHeaderReceived(const TDesC& aHeaderData) = 0;
    virtual void ClientBodyPartReceived(const TDesC8& aBodyPart) = 0;
    };

class CHttpClient : public CBase,
                    public MHTTPTransactionCallback,
                    public MHTTPAuthenticationCallback,
                    public MHTTPDataSupplier
    {
public:
    // Constructors and destructor
    ~CHttpClient();
    static CHttpClient* NewL(MHttpClientObserver& aObserver);
    static CHttpClient* NewLC(MHttpClientObserver& aObserver);

private:
    CHttpClient(MHttpClientObserver& aObserver);
    void ConstructL();

public:
    void StartHttpGetL(const TDesC8& aUri);
    void StartHttpPostL(const TDesC8& aUri,
                        const TDesC8& aContentType,
                        const TDesC8& aBody);
    void CancelTransactionL();
    TBool IsRunning();
    void SetCallBack( CMentorRecListView* aCallBack );
    void HandleErrorL(TInt aError);
```

```
private:
    void SetHeaderL(RHTTPHeaders aHeaders, TInt aHdrField,
                    const TDesC8& aHdrValue);

public:
    // MHTTPAuthenticationCallback
    TBool GetCredentialsL(const TUriC8 &aURI, RString aRealm,
                          RStringF aAuthenticationType,
                          RString &aUsername, RString &aPassword);

    // MHTTPTransactionCallback
    void MHFRunL(RHTTPTransaction aTransaction,
                 const THTTPEvent &aEvent);
    TInt MHFRunError(TInt aError, RHTTPTransaction aTransaction,
                     const THTTPEvent &aEvent);

    // MHTTPDataSupplier
    TBool GetNextDataPart(TPtrC8& aDataPart);
    void ReleaseData();
    TInt OverallDataSize();
    TInt Reset();

private:
    RHTTPSession iSession;
    RHTTPTransaction iTransaction;
    MHttpClientObserver& iObserver;
    CMentorRecListView* iView;
    TBool iRunning;

    HBufC8* iPostData;

    };

#endif // HTTPCLIENT_H
```

```
/*
 ============================================================================
 Name        : HttpClient.cpp
 Author      : kimmonii
 Description : CHttpClient implementation
 ============================================================================
 */

#include <aknnotewrappers.h>
#include <stringloader.h>
#include "HttpClient.h"
#include "MentorRec.pan"
#include "MentorRec_0xEC5A8700.rsg"
#include "MentorRecListView.h"

CHttpClient::CHttpClient(MHttpClientObserver& aObserver)
    : iObserver( aObserver ),
      iRunning( EFalse )
    {
    }

CHttpClient::~CHttpClient()
    {
    iSession.Close();
    }

CHttpClient* CHttpClient::NewLC(MHttpClientObserver& aObserver)
    {
    CHttpClient* self = new (ELeave) CHttpClient(aObserver);
    CleanupStack::PushL(self);
    self->ConstructL();
    return self;
    }

CHttpClient* CHttpClient::NewL(MHttpClientObserver& aObserver)
    {
    CHttpClient* self = CHttpClient::NewLC(aObserver);
    CleanupStack::Pop();
    return self;
    }

void CHttpClient::ConstructL()
    {
    iSession.OpenL();               // Uses default protocol: HTTP/TCP
    InstallAuthenticationL(iSession);
    }
```

```
void CHttpClient::StartHttpGetL(const TDesC8& aUri)
    {
    TUriParser8 uri;
    uri.Parse( aUri );

    RStringF method = iSession.StringPool().StringF(HTTP::EGET,
                                                    RHTTPSession::GetTable());
    iTransaction = iSession.OpenTransactionL(uri, *this, method);

    RHTTPHeaders headers = iTransaction.Request().GetHeaderCollection();
    SetHeaderL(headers, HTTP::EUserAgent, KUserAgent);
    SetHeaderL(headers, HTTP::EAccept, KAccept);

    iTransaction.SubmitL();

    iRunning = ETrue;

    HBufC* text = StringLoader::LoadLC(R_CONNECTING);
    iObserver.ClientEvent(*text);
    CleanupStack::PopAndDestroy(text);
    }

void CHttpClient::StartHttpPostL(const TDesC8& aUri,
                                 const TDesC8& aContentType,
                                 const TDesC8& aBody)
    {
    TUriParser8 uri;
    uri.Parse(aUri);

    delete iPostData;
    iPostData = NULL;
    iPostData = aBody.AllocL();

    RStringF method = iSession.StringPool().StringF(HTTP::EPOST,
                                                    RHTTPSession::GetTable());

    iTransaction = iSession.OpenTransactionL(uri, *this, method);

    RHTTPHeaders headers = iTransaction.Request().GetHeaderCollection();
    SetHeaderL(headers, HTTP::EUserAgent, KUserAgent);
    SetHeaderL(headers, HTTP::EAccept, KAccept);
    SetHeaderL(headers, HTTP::EContentType, aContentType);

    MHTTPDataSupplier* dataSupplier = this;
    iTransaction.Request().SetBody(*dataSupplier);

    iTransaction.SubmitL();

    iRunning = ETrue;
    HBufC* text = StringLoader::LoadLC(R_CONNECTING);
    iObserver.ClientEvent(*text);
    CleanupStack::PopAndDestroy(text);
    }
```

```
void CHttpClient::CancelTransactionL()
    {
    if (iRunning)
        {
        iTransaction.Close();
        iRunning = EFalse;

        HBufC* text = StringLoader::LoadLC(R_TRANSACTION_CANCELLED);
        iObserver.ClientEvent(*text);
        CleanupStack::PopAndDestroy(text);
        }
    }

TBool CHttpClient::IsRunning()
    {
    return iRunning;
    }

void CHttpClient::SetCallBack( CMentorRecListView* aCallBack )
    {
    iView = aCallBack;
    }

void CHttpClient::HandleErrorL(TInt aError)
    {
    HBufC* text = StringLoader::LoadLC(R_MHFRUN_ERROR);
    iObserver.ClientEvent(*text);
    CleanupStack::PopAndDestroy(text);

    // Remove wait dialog from view
    if ( iView )
        iView->RemoveWaitDialogL();
    }

void CHttpClient::SetHeaderL(RHTTPHeaders aHeaders, TInt aHdrField,
                             const TDesC8& aHdrValue)
    {
    RStringF valueStr = iSession.StringPool().OpenFStringL(aHdrValue);
    CleanupClosePushL(valueStr);
    THTTPHdrVal value(valueStr);
    RStringF field = iSession.StringPool().StringF(aHdrField,
                                                   RHTTPSession::GetTable());
    aHeaders.SetFieldL(field, value);
    CleanupStack::PopAndDestroy(&valueStr);
    }

TBool CHttpClient::GetCredentialsL(const TUriC8 &aURI, RString aRealm,
                                   RStringF aAuthenticationType,
                                   RString &aUsername, RString &aPassword)
    {
    return EFalse;
    }
```

```
void CHttpClient::MHFRunL(RHTTPTransaction aTransaction,
                          const THTTPEvent &aEvent)
    {
    switch (aEvent.iStatus)
        {
        case THTTPEvent::EGotResponseHeaders:
            {

            }
            break;
        case THTTPEvent::EGotResponseBodyData:
            {
            MHTTPDataSupplier* body = aTransaction.Response().Body();
            TPtrC8 dataChunk;
            TBool isLast = body->GetNextDataPart(dataChunk);
            iObserver.ClientBodyPartReceived(dataChunk);

            HBufC* text = StringLoader::LoadLC(R_BODY_PART_RECEIVED);
            iObserver.ClientEvent(*text);
            CleanupStack::PopAndDestroy(text);

            if (isLast)
                {
                HBufC* text = StringLoader::LoadLC(R_BODY_RECEIVED);
                iObserver.ClientEvent(*text);
                CleanupStack::PopAndDestroy(text);
                }
            body->ReleaseData();
            }
            break;
        case THTTPEvent::EResponseComplete:
            {
            HBufC* text = StringLoader::LoadLC(R_TRANSACTION_COMPLETED);
            iObserver.ClientEvent(*text);
            CleanupStack::PopAndDestroy(text);
            }
            break;
        case THTTPEvent::ESucceeded:
            {
            HBufC* text = StringLoader::LoadLC(R_TRANSACTION_SUCCEEDED);
            iObserver.ClientEvent(*text);
            CleanupStack::PopAndDestroy(text);

            // Remove wait dialog from view
            if ( iView )
                iView->RemoveWaitDialogL();
            }
            break;
        case THTTPEvent::EFailed:
            {
            HBufC* text = StringLoader::LoadLC(R_TRANSACTION_FAILED);
            iObserver.ClientEvent(*text);
            CleanupStack::PopAndDestroy(text);

            // Remove wait dialog from view
```

```
            if ( iView )
                iView->RemoveWaitDialogL();

            aTransaction.Close();
            iRunning = EFalse;
            }
            break;
        default:
            {
            HBufC* text;
            if (aEvent.iStatus < 0)
                {
                text = StringLoader::LoadLC(R_TRANSACTION_ERROR);
                // Remove wait dialog from view
                if ( iView )
                    iView->RemoveWaitDialogL();

                aTransaction.Close();
                iRunning = EFalse;
                }
            else
                {
                text = StringLoader::LoadLC(R_TRANSACTION_UNKNOWN_EVENT);
                }

            iObserver.ClientEvent(*text);
            CleanupStack::PopAndDestroy(text);
            }
            break;
        }
    }

TInt CHttpClient::MHFRunError(TInt aError, RHTTPTransaction aTransaction,
                                        const THTTPEvent &aEvent)
    {
    TRAPD(error, HandleErrorL(aError));
    if (error)
        Panic(EMentorRecHttpClient);
    return KErrNone;
    }

TBool CHttpClient::GetNextDataPart(TPtrC8& aDataPart)
    {
    if(iPostData)
        {
        // Give a pointer to next data part, return ETrue if no more parts
        aDataPart.Set(iPostData->Des());
        }
    return ETrue;
    }
```

```
void CHttpClient::ReleaseData()
    {
    delete iPostData;
    iPostData = NULL;
    }

TInt CHttpClient::OverallDataSize()
    {
    if(iPostData)
        return iPostData->Length();
    else
        return KErrNotFound ;
    }

TInt CHttpClient::Reset()
    {
    return KErrNone;
    }
```

```
/*
 ============================================================================
 Name        : CameraEngine.h
 Author      : kimmonii
 Description : Declares CCameraEngine and MCameraEngineObserver classes.
 ============================================================================
 */
#ifndef CAMERAENGINE_H
#define CAMERAENGINE_H

// INCLUDE FILES
#include <e32std.h>
#include <e32base.h>
#include <ecam.h>

class MCameraEngineObserver
    {
public:
    virtual void OnViewFinderFrameReady(CFbsBitmap& aBitmap) = 0;
    virtual void OnImageSaved() = 0;
    virtual void OnError(TInt aError) = 0;
    };

class CCameraEngine : public CBase, public MCameraObserver
    {
private:
    enum TViewFinderMode
        {
        EViewFinderNone,
        EViewFinderBitmap,
        EViewFinderDirect
        };

public:
    // Constructors and destructor
    static CCameraEngine* NewL(MCameraEngineObserver& aObserver,
                               const CCoeControl& aControl);
    static CCameraEngine* NewLC(MCameraEngineObserver& aObserver,
                                const CCoeControl& aControl);
    ~CCameraEngine();

private:
    // Constructors
    CCameraEngine(MCameraEngineObserver& aObserver,
                  const CCoeControl& aControl);
    void ConstructL();

public:
    static TBool IsCameraAvailable();
    TInt CameraHandle();
    TUint32 SupportedImageFormats();
    TPtrC FileName();
    void DisplayViewFinderL();
    void StopViewFinderL();
    void CaptureImageL(const TDesC& aFileName);
```

```
private:
    // From MCameraObserver
    void ReserveComplete(TInt aError);
    void PowerOnComplete(TInt aError);
    void ViewFinderFrameReady(CFbsBitmap &aFrame);
    void ImageReady(CFbsBitmap *aBitmap, HBufC8 *aData, TInt aError);
    void FrameBufferReady(MFrameBuffer *aFrameBuffer, TInt aError);

private:
    // Private methods
    void DoPrepareImageCapture();
    void DoSaveImageL(const TDesC8& aData);

private:
    // Data members
    MCameraEngineObserver& iObserver;
    const CCoeControl& iControl;

    CCamera* iCamera;
    TCameraInfo iCameraInfo;

    TViewFinderMode iViewFinderMode;
    TSize iViewFinderSize;
    TBool iIsCaptureReady;
    RBuf iFileName;
    };

#endif // CAMERAENGINE_H
```

```
/*
 ============================================================================
 Name        : CameraEngine.cpp
 Author      : kimmonii
 Description : CCameraEngine implementation
 ============================================================================
 */

#include <coecntrl.h>
#include "CameraEngine.h"

CCameraEngine::CCameraEngine(MCameraEngineObserver& aObserver,
                             const CCoeControl& aControl)
    : iObserver(aObserver),
      iControl(aControl)
    {
    }

CCameraEngine::~CCameraEngine()
    {
    if (iCamera)
        {
        iCamera->Release();
        delete iCamera;
        }
    iFileName.Close();
    }

CCameraEngine* CCameraEngine::NewLC(MCameraEngineObserver& aObserver,
                                    const CCoeControl& aControl)
    {
    CCameraEngine* self = new (ELeave) CCameraEngine(aObserver, aControl);
    CleanupStack::PushL(self);
    self->ConstructL();
    return self;
    }

CCameraEngine* CCameraEngine::NewL(MCameraEngineObserver& aObserver,
                                   const CCoeControl& aControl)
    {
    CCameraEngine* self = CCameraEngine::NewLC(aObserver, aControl);
    CleanupStack::Pop(self);
    return self;
    }
```

```
void CCameraEngine::ConstructL()
    {
    CCamera* cam;
    TCameraInfo camInfo;
    for (TInt i = 0 ; i < CCamera::CamerasAvailable() ; i++)
        {
        cam = CCamera::NewL(*this, i);
        cam->CameraInfo(camInfo);
        if (camInfo.iOrientation == TCameraInfo::EOrientationOutwards)
            {
            iCamera = cam;
            iCameraInfo = camInfo;

            // Select viewfinder mode
            if ( ( iCameraInfo.iOptionsSupported &
                   TCameraInfo::EViewFinderDirectSupported ) > 0)
                {
                iViewFinderMode = EViewFinderDirect;
                }
            else if ( ( iCameraInfo.iOptionsSupported &
                        TCameraInfo::EViewFinderBitmapsSupported ) > 0)
                {
                iViewFinderMode = EViewFinderBitmap;
                }
            else
                {
                iViewFinderMode = EViewFinderNone;
                }

            return;
            }
        else
            {
            delete cam;
            }
        }
    }

TBool CCameraEngine::IsCameraAvailable()
    {
    return (CCamera::CamerasAvailable() > 0);
    }

TInt CCameraEngine::CameraHandle()
    {
    return iCamera->Handle();
    }

TUint32 CCameraEngine::SupportedImageFormats()
    {
    return iCameraInfo.iImageFormatsSupported;
    }
```

```
TPtrC CCameraEngine::FileName()
    {
    return iFileName;
    }

void CCameraEngine::DisplayViewFinderL()
    {
    if (!iCamera) // No camera available
        {
        User::Leave(KErrNotFound);
        }

    if (iViewFinderMode == EViewFinderNone) // VievFinder not supported
        {
        iObserver.OnError(KErrNotSupported);
        }
    else
        {
        iViewFinderSize = TSize(320, 280); // iControl.Size() does not work
        iCamera->Reserve();
        }
    }

void CCameraEngine::StopViewFinderL()
    {
    if (!iCamera) // No camera available
        {
        User::Leave(KErrNotFound);
        }

    iCamera->StopViewFinder();
    iCamera->Release();

    iIsCaptureReady = EFalse;
    }

void CCameraEngine::CaptureImageL(const TDesC& aFileName)
    {
    if (!iCamera) // No camera available
        {
        User::Leave(KErrNotFound);
        }

    if (!iIsCaptureReady) // Not ready to capture
        {
        User::Leave(KErrNotReady);
        }

    iFileName.ReAllocL(aFileName.Length());
    iFileName.Copy(aFileName);

    iCamera->CaptureImage();
    }

void CCameraEngine::ReserveComplete(TInt aError)
```

```
    {
    if (aError == KErrNone)
        {
        iCamera->PowerOn();
        }
    else
        {
        iObserver.OnError(aError);
        }
    }

void CCameraEngine::PowerOnComplete(TInt aError)
    {
    if (aError == KErrNone)
        {
        TInt error = KErrNone;

        if (iViewFinderMode == EViewFinderDirect)
            { // Start direct viewFinder
            TRect rect(0, 0, iViewFinderSize.iWidth, iViewFinderSize.iHeight);
            TRAP(error,
                iCamera->StartViewFinderDirectL(
                                iControl.ControlEnv()->WsSession(),
                                *iControl.ControlEnv()->ScreenDevice(),
                                *iControl.DrawableWindow(),
                                rect )
                );
            }
        else
            { // Start bitmap viewFinder
            TRAP(error, iCamera->StartViewFinderBitmapsL(iViewFinderSize));
            }

        if (error == KErrNone)
            {
            DoPrepareImageCapture();
            }
        else
            {
            iObserver.OnError(error);
            }
        }
    else
        {
        iCamera->Release();
        iObserver.OnError(aError);
        }

    }
```

```
void CCameraEngine::ViewFinderFrameReady(CFbsBitmap& aFrame)
    {
    // Reset the inactivity timer to keep the light on
    User::ResetInactivityTime();

    iObserver.OnViewFinderFrameReady(aFrame);
    }

void CCameraEngine::ImageReady(CFbsBitmap* /*aBitmap*/, HBufC8* aData,
                               TInt aError)
    {
    if ( (aError == KErrNone) && aData )
        {
        TRAPD(error, DoSaveImageL(*aData));
        if (error == KErrNone)
            {
            iObserver.OnImageSaved();
            }
        else
            {
            iObserver.OnError(error);
            }
        delete aData;
        }
    else
        {
        iObserver.OnError(aError);
        }

    }

void CCameraEngine::FrameBufferReady(MFrameBuffer* /*aFrameBuffer*/, TInt
/*aError*/)
    {
    }
```

```
void CCameraEngine::DoPrepareImageCapture()
    {
    CCamera::TFormat format = CCamera::EFormatUserDefined;
    if ( (iCameraInfo.iImageFormatsSupported & CCamera::EFormatJpeg) > 0 )
        {
        format = CCamera::EFormatJpeg;
        }
    else
        {
        iObserver.OnError(KErrNotSupported);
        return;
        }

    TRAPD(error, iCamera->PrepareImageCaptureL(format, 0) );
    if (error == KErrNone)
        {
        iIsCaptureReady = ETrue;
        }
    else
        {
        iObserver.OnError(error);
        }

    }

void CCameraEngine::DoSaveImageL(const TDesC8& aData)
    {
    RFile file;
    User::LeaveIfError(file.Replace(iControl.ControlEnv()->FsSession(),
                                    iFileName,
                                    EFileWrite));
    CleanupClosePushL(file);
    User::LeaveIfError(file.Write(aData));
    CleanupStack::PopAndDestroy(&file);
    }
```

```
/*
 ============================================================================
 Name        : MentorVideoRecorder.h
 Author      : kimmonii
 Description : CMentorVideoRecorder declaration
 ============================================================================
 */
#ifndef MENTORVIDEORECORDER_H
#define MENTORVIDEORECORDER_H

// INCLUDES
#include <e32std.h>
#include <e32base.h>
#include <videorecorder.h>

// CONSTANTS
const TInt KKiloByte = 1024;
const TInt KMaxVideoSize = 1024 * KKiloByte;
_LIT(KMp4FileExtension, ".mp4");

class CMentorVideoRecorder : public CBase,
                             public MVideoRecorderUtilityObserver
    {
public:
    ~CMentorVideoRecorder();
    static CMentorVideoRecorder* NewL();
    static CMentorVideoRecorder* NewLC();

private:
    CMentorVideoRecorder();
    void ConstructL();

public:
    void RecordL(const TDesC& aFileName, TInt aCameraHandle);
    void Stop();
    TPtrC8 MimeType();

public:
    // From MVideoRecorderUtilityObserver
    void MvruoOpenComplete(TInt aError);
    void MvruoPrepareComplete(TInt aError);
    void MvruoRecordComplete(TInt aError);
    void MvruoEvent(const TMMFEvent &aEvent);

public:
    void DisplayErrorMessage(TInt aError);

private:
    CVideoRecorderUtility* iVideoRecorderUtility;
    TUid iControllerUid;
    TUid iFormatUid;
    RBuf8 iMimeType;

    };
#endif // MENTORVIDEORECORDER_H
```

```
/*
 ============================================================================
 Name        : Video.h
 Author      : kimmonii
 Description : CMentorVideoRecorder implementation
 ============================================================================
 */

#include <eikenv.h>
#include <aknnotewrappers.h>
#include "MentorVideoRecorder.h"

CMentorVideoRecorder::CMentorVideoRecorder()
    {
    }

CMentorVideoRecorder::~CMentorVideoRecorder()
    {
    if (iVideoRecorderUtility)
        {
        iVideoRecorderUtility->Close();
        }
    delete iVideoRecorderUtility;
    iMimeType.Close();
    }

CMentorVideoRecorder* CMentorVideoRecorder::NewLC()
    {
    CMentorVideoRecorder* self = new (ELeave) CMentorVideoRecorder();
    CleanupStack::PushL(self);
    self->ConstructL();
    return self;
    }

CMentorVideoRecorder* CMentorVideoRecorder::NewL()
    {
    CMentorVideoRecorder* self = CMentorVideoRecorder::NewLC();
    CleanupStack::Pop(); // self;
    return self;
    }

void CMentorVideoRecorder::ConstructL()
    {
    iVideoRecorderUtility = CVideoRecorderUtility::NewL(*this);

    CMMFControllerPluginSelectionParameters* controllerSelect =
                        CMMFControllerPluginSelectionParameters::NewLC();
    CMMFFormatSelectionParameters* formatSelect =
                        CMMFFormatSelectionParameters::NewLC();

    // Set the record format
    formatSelect->SetMatchToFileNameL(KMp4FileExtension);
    controllerSelect->SetRequiredRecordFormatSupportL(*formatSelect);

    // Set media ids to video with audio
```

```
RArray<TUid> mediaIds;
CleanupClosePushL(mediaIds);
User::LeaveIfError(mediaIds.Append(KUidMediaTypeVideo));
User::LeaveIfError(mediaIds.Append(KUidMediaTypeAudio));
controllerSelect->SetMediaIdsL(mediaIds,
                CMMFPluginSelectionParameters::EAllowOtherMediaIds);

// No preferred controller supplier
controllerSelect->SetPreferredSupplierL(KNullDesC,
                CMMFPluginSelectionParameters::ENoPreferredSupplierMatch);

RMMFControllerImplInfoArray controllers;
CleanupResetAndDestroyPushL(controllers);
controllerSelect->ListImplementationsL(controllers);

// Get a controller with record formats
iControllerUid = KNullUid;
iFormatUid = KNullUid;
for (TInt i = 0 ; i < controllers.Count() ; i++)
    {
    RMMFFormatImplInfoArray recFormats = controllers[i]->RecordFormats();
    for (TInt j = 0 ; j < recFormats.Count() ; j++)
        {
        iControllerUid = controllers[i]->Uid();

        const CDesC8Array& mimeTypes= recFormats[j]->SupportedMimeTypes();

        if (mimeTypes.Count() > 0)
            {
            iMimeType.CreateL(mimeTypes[0]);

            RBuf text;
            CleanupClosePushL(text);
            text.CreateL(mimeTypes[0].Length());
            text.Copy(mimeTypes[0]);

            CAknInformationNote* note = new(ELeave) CAknInformationNote;
            note->ExecuteLD(text);

            CleanupStack::PopAndDestroy(&text);

            break;
            }
        }
    }
CleanupStack::PopAndDestroy(&controllers);
CleanupStack::PopAndDestroy(&mediaIds);
CleanupStack::PopAndDestroy(formatSelect);
CleanupStack::PopAndDestroy(controllerSelect);

if (iControllerUid == KNullUid) // No suitable controller found
    {
    User::Leave(KErrNotSupported);
    }
}
```

```
void CMentorVideoRecorder::RecordL(const TDesC& aFileName, TInt aCameraHandle)
    {
    iVideoRecorderUtility->Close();
    if (iVideoRecorderUtility)
        {
        iVideoRecorderUtility->OpenFileL(aFileName, aCameraHandle,
                                         iControllerUid, iFormatUid);
        }
    }

void CMentorVideoRecorder::Stop()
    {
    iVideoRecorderUtility->Stop();
    iVideoRecorderUtility->Close();
    }

TPtrC8 CMentorVideoRecorder::MimeType()
    {
    return iMimeType;
    }

void CMentorVideoRecorder::MvruoOpenComplete(TInt aError)
    {
    if (aError == KErrNone)
        {
        TRAPD(error,
            iVideoRecorderUtility->SetMaxClipSizeL(KMaxVideoSize);
            iVideoRecorderUtility->SetAudioEnabledL(ETrue);
            );
        if (aError == KErrNone)
            {
            iVideoRecorderUtility->Prepare();
            }
        else
            {
            DisplayErrorMessage(aError);
            }
        }
    else
        {
        DisplayErrorMessage(aError);
        }
    }

void CMentorVideoRecorder::MvruoPrepareComplete(TInt aError)
    {
    if (aError == KErrNone)
        {
        iVideoRecorderUtility->Record();
        }
    else
        {
        DisplayErrorMessage(aError);
        }
    }
```

```
void CMentorVideoRecorder::MvruoRecordComplete(TInt aError)
    {
    Stop();
    }

void CMentorVideoRecorder::MvruoEvent(const TMMFEvent &aEvent)
    {

    }

void CMentorVideoRecorder::DisplayErrorMessage(TInt aError)
    {
    const TInt KMaxBuffer = 15;
    _LIT(KErrorMessage, "Error: %d");
    TBuf<KMaxBuffer> buffer;
    buffer.AppendFormat(KErrorMessage, aError);
    TRAP_IGNORE(CEikonEnv::Static()->InfoWinL(KNullDesC, buffer));
    }
```

Kuva. Päänäkymä, jossa videolista

Kuva. Päänäkymän valikko esillä

Kuva. Videon tietojen muokkaus

Kuva. Kuvausnäkymä, jossa näkyvissä kameran etsin